Ａｄｖａｎｃｅ ＣＡＤ

プログラミングマニュアル

Advance CAD software version 21

プログラミングマニュアル

Advance CAD software version 20

2012 年　11 月 2 日 第 1 版
2016 年　 9 月 29 日 第 2 版

〒 141-8522 東京都品川区大崎１－２－２ アートヴィレッジ大崎 セントラルタワー

● 技術的なお問い合わせ先

Advance CAD の技術的なご質問は下記で受付けております。
Advance CAD ソフトウェア保守契約に加入されているお客様に限らせていただきます。

--------- Advance CAD ホットラインサービス ----------
電話番号　：　03-5434-0095
FAX 番号　：　03-5434-0056
E-mail　：　acad_support@ctc-g.co.jp
---------- 受付時間 ： 平日 9:00 ～ 17:30 --

●

目次

第 1 章　はじめに

1.1　プログラミングインターフェイス

アプリケーションの各種パラメータ設定ファイルを変更する、メニューを変更する、マクロプログラミングを使うなどの方法で、ユーザが使い易いようにアプリケーションをカスタマイズできます。これらはアプリケーションが提供する汎用機能をユーザの実際の使い方に適合するようにし、より効果的な使い方を実現します。
もうひとつのカスタマイズ方法はアプリケーションにユーザ独自の機能を追加し、アプリケーションの機能を拡張することです。ユーザが、独自機能をプログラミングしてアプリケーションに組み込み、コマンドとして使う方法です。これを**プログラミングインターフェイス**と呼びます。
ユーザコマンドの追加方法はアプリケーションが提供するコマンドの実装方法とほとんど同じです。プログラミングインターフェイスのために特別に追加したものではありません。ユーザコマンドはアプリケーションのコマンドと同等です。
最初にユーザコマンドの名前を追加します。次にこのコマンドの機能を実現する関数をプログラミングします。この関数を**コマンドハンドラ**と呼びます。このコマンドハンドラのプログラミングの説明が本書の主目的です。続いてコマンドハンドラをプログラミングしたソースコードをコンパイルし、ユーザライブラリを再作成します。アプリケーション はユーザライブラリを動的リンクしていますので、アプリケーションを起動すれば、すぐにユーザコマンドをテストできます。詳しくは『AppendixA ユーザライブラリのビルド』を参照してください。

1.2　プログラミング言語

アプリケーション開発のプログラミング言語は C++ です。バージョン 18 から C++ に変更しました。しかしこれまでは、プログラミングインターフェイスでは C++ の C 言語互換部分だけを使用したプログラミングに限定し、大きな変更を行わないようにしてきました。バージョン 20 では、プログラミングインターフェイスにも C++ の**オブジェクト指向プログラミング**を導入します。クラスや継承などの C++ のオブジェクト指向の機能を使用します。
また、C++ **標準テンプレートライブラリ**（STL：Standard Template Laibrary）を使用しています。たとえば、文字列は char 配列ではなく、標準テンプレートライブラリの std::string を使用している箇所が多々あります。std::string は任意長の文字列を表現でき、長さが固定の char 配列にはない便利さと安全性を備えています。便利に使えるクラステンプレートが沢山ありますが、必ず使用するのは std::string です。
本書では C++ 言語、オブジェクト指向プログラミング、標準テンプレートライブラリの説明は行いません。

1.3 主な変更点

「プログラミング言語」の節で簡単に触れましたが、プログラミングインターフェイスの最も大きな変更はオブジェクト指向プログラミングになったことです。簡単にいえば、今までの関数に代わってオブジェクトのメソッド使ってプログラミングすることです。
これらの変更のため本書の内容を大幅に改訂しました。アプリケーションの関数の説明の多くの部分はそれに代わるクラスの解説に書き換えました。本書で解説するクラスは今までの関数をカバーする範囲です。クラスが持つメソッドの内、今までの関数と同等以上の機能を実現するのに必要なメソッドを主に説明します。説明のないメソッドは将来変更するか削除する可能性があるメソッドです。それからアプリケーションの特殊な用途のメソッドも説明がありません。

第 2 章はコマンドハンドラのプログラミングの方法と基礎的なテクニックを説明します。コマンドの追加方法は今までと同じです。

第 7 章はコマンドハンドラの実装に使用する種々の関数を説明します。すこし変更があります。ひとつはドラッギングするデータの登録方法を変更しました。今までは、ドラッギングデータを登録するときにはアイテムや図形表示関数を使用していました。表示関数が図形を表示する代わりにドラッギングデータ領域へ保存することを行っていました。今後はドラッギングデータ登録機能を持つクラスを使います。表示とドラッギングデータ登録は完全に分離しました。
もうひとつは、「アクティブリスト」と「ハイライトリスト」はクラスを使った実装になったことです。

第 3 章はアイテムの参照方法、アイテムの作成・変更について概要を説明します。ここはオブジェクト指向プログラミングになり大きく変更しました。新しいアプリケーションデータの操作方法の全体像を理解してもらうための章です。
アプリケーションの主要なデータであるモデルは C++ のクラスを使った実装になりました。モデルが持つアイテムをはじめとする各種データも全てクラスを使った実装です。アイテムのデータを取り出すことを考えてみましょう。以前はアイテムに対して open()、read()、close() といった関数を使って行いました。最初に open() 関数で指定したアイテムのデータ取得の準備をします。これが成功したら、readhead()、readdata() 関数を使ってアイテムのサブレコードを取得します。アイテムの先頭から順次サブレコード取得を繰り返します。readhead() 関数はサブレコードの情報を引数の ItmSR 構造体にコピイし、readdata() 関数はサブレコードの内容を引数の配列にコピイします。データ取得を終える時は必ず close() 関数を呼び出さなければなりません。同時にオープンできるアイテム数には制限があるので、close() 関数で開放する必要がありました。
これがオブジェクト指向プログラミングでは次のようになります。まずデータを取り出したいアイテムオブジェクトを取得します。アイテムオブジェクトを得たら、オブジェクトのメソッドを使ってサブレコードを取得します。サブレコードは先頭から順次取得することもできますが、任意の順番で取得することができます。取得したサブレコードもオブジェクトですから、サブレコードの内容を配列にコピイする必要がありません。サブレコードオブジェクトのメソッドを使ってサブレコードの内容にアクセスする方法になり、簡単で安全な方法が使用できるようになりました。オブジェクト指向プログラミングでは ItmSR 構造体は使用しなくなり、サブレコードを表現するクラスを使用します（第 6 章）。
アイテムを作成・変更するのに使うテンポラリアイテムの操作も同様に変更されました（第 4 章）。
以降の章で詳細を解説します。

第 4 章はアプリケーションのデータを管理する最上位のモデルクラスを解説します。そしてパーマネントアイテムを保持するクラス、パーマネントアイテムを作成／編集するためのテンポラリアイテムのクラスを解説しています。

第 5 章はアイテムのデータ構造を解説します。アイテムの種類ごとに保持するサブレコードの種類、その順序を記述しています。この規則から外れたアイテムを作ってしまうと問題があります。この章の内容はオブジェクト指向とは関係ありません。

第 6 章はアイテムを構成する最小単位であるサブレコードを解説します。サブレコードはその種類ごとにクラスで実装します。サブレコードのデータはクラスのメソッドを使って設定／取得します。以前は、サブレコードデータを配列にコピイし、それに変更を施し、変更したデータをサブレコードに書き

込むといった方法を行っていました。これは、サブレコードデータのレイアウトに依存したプログラムを書くことになり、バージョンアップでレイアウトが変更になったときの対応が困難であるという問題がありました。サブレコードクラスはデータレイアウトをクラス内部に隠蔽することでこのような問題を回避します。

第 8 章から第 10 章はモデルクラスが管理するデータを解説します。
第 8 章は幾何演算定数、製図定数、モデルタイトルなどを解説します。
第 9 章はピクチャを解説します。
第 10 章はドローイングレイアウトを解説します。

第 11 章は幾何クラスを解説します。
以前、幾何図形は DGEOM、DPOINT、DLINE、DARC、DBZC などの構造体で表現していました。これらの幾何図形も幾何クラスになりました。そして交点計算や接線計算など幾何演算の関数は幾何クラスのメソッドになりました。

第 12 章から第 14 章はアイテムに関する解説をします。
第 12 章は曲線アイテムの交点を計算するなどの曲線アイテム操作を中心に解説します。
第 13 章は注記アイテム、寸法アイテムの作成方法を中心に解説します。
第 14 章はコンポジットアイテム、シンボルやサブモデルなどの構造化アイテムを解説します。

第 17 章はユーザ定義定数を解説します。基本的な仕組みに変更ありませんが、少し変更があります。ユーザ定義定数をモデルファイルに保存する場合に呼び出されるフック関数名を変更しました。以前の comusrio() 関数は廃止し、それに代わる新しい三つの関数 comusrsize()、comusrread()、comusrwrite() で置き換えました。また、モデルファイルへの入出力を行う関数 MdlfRead()、MdlfWrite() は廃止し、代わりに AwUserIO クラスを使用するようになりました。

1.4 文字コード

文字コードは日本語 EUC(EUC-JP) に統一しています。プログラム中の文字列（メッセージやエラーメッセージを含め）はすべて日本語 EUC、マルチバイト文字列です。モデルファイル、シンボルファイルなどのデータファイル中の文字列も同様です。

Windows の場合、文字コードは MS 漢字コード (またはシフト JIS) です。しかしソースコードに直接 MS 漢字コードの文字定数を記述してはいけません。プログラムで使用する文字列は、全てメッセージファイルに記述しておき、それを使うべきです。メニュー、メッセージ、エラーメッセージファイルは MS 漢字コードで記述します。

止むを得ずソースコードに MS 漢字コードを直接記述する場合、AwEucEncoder クラスのメソッドを使って日本語 EUC に変換します。

```
std::string eucstr = AwEucEncoder::Sjis2Euc("日本語の文字列");
Mesagdisp(MZONECOLOR1, 1, 1, 0, 10 * eucstr.length(), eucstr.c_str());
```

またテキストファイルから日本語文字列を読み込んだ場合も、MS 漢字コードを日本語 EUC に変換しなければなりません。ファイルに出力する場合は逆の変換をしなければなりません。この変換を行ってくれるのが AwFileIO クラスです。
テキストファイルを 1 行読み込み表示するには次のようにします。AwFileIO オブジェクトの Fgets メソッドで必要な変換が行われます。

```
char text[80];
AwFileIO file;
if (! file.Fopen("myfile.txt", "r"))
  return;
if (file.Fgets(text, sizeof(text)) == NULL)
  return;
file.Fclose();
```

```
Mesagdisp(MZONECOLOR1, 1, 1, 0, 10 * strlen(text), text);
```

このように プログラムの文字列はすべて日本語 EUC でなければなりません。誤って日本語 EUC 以外の文字コードが混入すると、モデルファイル、シンボルファイルなどの重要なデータファイル中の文字列は信頼出来ず、正しい結果が得られませんので注意してください。

1.5 ファイルパス

ディレクトリ区切りはスラッシュ ('/') を使用します。文字コードは日本語 EUC です。
Windows のファイルパスはプログラム上ではバックスラッシュをスラッシュに置き換えたものにします。

```
C:/acad/files/SAMPLE.MDL
```

Windows ファイルをオープンするなど実際にファイル操作を行う時点で、プラットホーム依存のパスに変換します。AwFile クラスのメソッドが必要な変換を行います。

1.6 プログラミングスタイル

次章からはアプリケーションの関数、クラスの解説がでてきます。その前に本書のプログラミングスタイルを説明します。例題プログラムなどの理解に役立つでしょう。

1.6.1 定数

アイテムタイプやサブレコードタイプなどを表す定数はクラスヘッダーファイル中に enum で定義します。例えば、AwItem::LINE は AwItem クラスで定義している線分アイテムの番号です。AwSr::LINE は AwSr クラスで定義している線分サブレコードの番号です。
「クラス名 :: 定数名」でおおよそ推測できると思われますので、例題プログラムでは断わりなくこれらの定数を使っています。

1.6.2 メソッドの引数、戻り値

配列引数は配列であることを強調するため、ポインタ形式ではなく [] を使います (const T[])。文字列は文字配列ですが、習慣に従いポインタ形式 (const char*) を使います。
文字列は char 配列よりも標準テンプレートライブラリの std::string を使用するようにしています。

入力引数は const 参照です (const T&)。引数の値変更を防ぐため const を付け、効率のため値渡しではなく参照にします。ポインタ引数は null ポインタでないか確認する必要がありますが、参照はそのようなことが不要という利点があります。入力引数がオプションで null ポインタを許すときなどは const ポインタとします (const T*)。

出力引数は出力引数であることを視覚的に強調するためにポインタ形式を使用します (T*)。

メソッドが返すデータの型が基本データ型 (int, double, bool など）であれば値返しです。
メソッドがオブジェクトのメンバー変数の参照やポインタを返す場合 const と non-const を使い分けます。メンバー変数がオブジェクトのときは効率のため const 参照 (const T&) を返します。メンバー変数がポインタ型のときは null ポインタを返すことがあるので const ポインタ (const T*) を返します。オブジェクトの値の変更を許すときは non-const ポインタを返します。いずれの場合も、メソッドの呼び出し側は得たオブジェクトを削除 (delete) してはいけません。

メソッドが新しくオブジェクトを生成 (new) して返すなら non-const ポインタです。このような場合メソッドの呼び出し側がオブジェクトの所有権を持つことが多く、呼び出し側はそのオブジェクトを削除

(delete) しなければなりません。呼び出し側がオブジェクトの所有権を持つ場合はメソッドの解説で明示します。

1.6.3 const メソッド

オブジェクトの状態を変更しないメソッドは const メソッドとします。メンバー変数の const ポインタを返すメソッドは const メソッドです。外部からは、このメソッドで得た const ポインタを使ってオブジェクトの const メソッドを使うことはできますが、オブジェクトの状態を変更する non-const メソッドを使うことはできません。これに対してメンバー変数の non-const ポインタを返すメソッドは non-const メソッドです。このメソッド自身はオブジェクトの状態を変更しませんが、外部からこの non-const ポインタを使ってオブジェクトの状態を変更する non-const メソッドを使うことができます。メンバー変数のオブジェクトの状態を変更する手段を与えるので non-const メソッドです。

```
class Sample {
  public:
    Sample() : m_t() {}
  public:
    const T& Get() const { return m_t; } // const メソッド
    T* Get() { return &m_t; }            // non-const メソッド
  private:
    T m_t;
};
```

1.6.4 static_cast

キャストは C-style ではなく static_cast を使用します。AwSrLine クラスが AwSr クラスを継承していて、AwSr ポインタを AwSrLine ポインタにキャストするには次のように記述します。

```
const AwSr* pSr = item.GetSrAt(i);
if (pSr->GetId() == AwSr::LINE)
  const AwSrLine* pSrLine = static_cast<const AwSrLine*>(pSr);
```

次の例はコンパイルエラーになります。

```
 AwSrLine* pSrLine = static_cast<AwSrLine*>(pSr);
```

const ポインタ pSr を non-const ポインタ pSrLine に変換しようとするからです。通常 const ポインタ を non-const ポインタに強制的に変換するのは誤りです。それをコンパイラーが検出してくれます。

static_cast はクラスの継承関係を調べキャスト可能かどうか判定します。キャストできないと判断したときはコンパイルエラーになります。static_cast の名前はコンパイル時に静的な型検査を行うことに由来します。コンパイラは継承関係を調べるためにクラス定義（インクルードファイル）を調べます。上記の例では次のインクルードが必要です。

```
#include "AwSrLine.h"
```

一方 C-style キャストでは const を忘れてもコンパイルが成功します。

```
AwSrLine* pSrLine = (AwSrLine*)pSr;
```

C-style キャストは型検査をしないので次のような誤ったコードも書けてしまいます。

```
G2Line* pLine = (G2Line*)pSrLine;
```

両方ともコンパイルは成功しますが、実行時に問題を起こすでしょう。コンパイラは C-style キャストに対しては検査しないので、誤りに気づかないことが多く危険です。

1.6.5 std::auto_ptr

オブジェクトを生成（new）したら、オブジェクトが不要になった時点で削除（delete）しなければなりません。

```
G2Point* pPoint = new G2Point(); // オブジェクトを生成
if (pPoint == NULL)
```

```
    return;
  pPoint->Set(30.0, 40.0);
  delete pPoint; // オブジェクトを削除
  return;
```

オブジェクトの削除を忘れるとメモリリークが発生し、利用可能なメモリが枯渇します。オブジェクトの削除を確実にするには標準テンプレートライブラリの std::auto_ptr を使用するとよいでしょう。

```
  std::auto_ptr<G2Point> apPoint(new G2Point()); // オブジェクトを生成
  if (apPoint.get() == NULL)
    return;
  apPoint->Set(30.0, 40.0);
  return;
```

apPoint 変数がスコープを抜けるとき自動的にデストラクタが呼び出され apPoint 変数が持っている G2Point オブジェクトを削除します。
apPoint 変数に別の G2Point オブジェクトを設定するには次のようにします。apPoint 変数は現在持っているオブジェクトを削除し、引数で渡されたオブジェクトを保持します。

```
  apPoint.reset(new G2Point(50.0, 60.0));
```

apPoint 変数が持っている G2Point オブジェクトを明示的に削除するには次のようにします。

```
  apPoint.reset();
```

1.6.6 copier

コピイコンストラクタと代入演算子 (=) はオブジェクトをコピイします。この二つを copier と呼びます。
クラスがポインタ型のメンバ変数を持つなら、そのオブジェクトのコピイは注意しなければなりません。コンパイラが自動生成するデフォルトコンストラクタと代入演算子は、ポインタ型変数に対して単純に変数が指すアドレスをコピイするだけなので、コピイ元とコピイ先のふたつのオブジェクトのポインタ型変数が同じアドレスを指すことになります。これを浅いコピイ (shallow copy) といいます。通常これではうまくないので、ポインタ型変数の指すオブジェクトに対してはコピイを生成 (new) するようにします。これを深いコピイ (deep copy) といいます。深いコピイでは、ポインタ型変数の指すオブジェクトがさらにポインタ型メンバ変数を持っていると再帰的にコピイを行うことになります。アプリケーションのデータを持つオブジェクトなどは膨大な数のオブジェクトを持つことがあり、このコピイによりメモリ不足になる可能性があります。

オブジェクトのコピイを禁止するため copier を使えないようにすることがあります。次の例のように copier を private にしたクラスはオブジェクトのコピイを禁止しています。

```
  class AwModel {
  public:
    AwModel();
   ~AwModel();
  private:
    AwModel(const AwModel& model);
    AwModel& operator=(const AwModel& model);
  };
```

次の例はモデルオブジェクトのコピイを作ろうとしています。

```
  AwModel model = *pModel; // コピイコンストラクタ AwModel model(*pModel)
```

このステートメントは、AwModel クラスのオブジェクトがコピイ禁止になっているため、コンパイル時にエラーになります。

コピイ禁止クラスのオブジェクトをメソッドの引数とするときは値渡しには出来ません。このような引数は参照かポインタにしなければなりません。

第 2 章　コマンドハンドラの基本

この章ではユーザコマンド追加方法を下記の手順にそって説明します。
(1)　**コマンド名を追加します。**
(2)　**コマンド名に対応するコマンドハンドラの呼び出し経路をプログラミングします。**
(3)　**コマンドハンドラをプログラミングします。**

コマンドハンドラが受け取るユーザの入力を処理する方法、ユーザの入力データを表す TOKEN 構造体、メッセージの表示方法などコマンドハンドラのプログラミングの基本を説明します。

2.1　ユーザコマンドの登録

ユーザコマンドは（menu ディレクトリにある）USERCMD.MEN に追加します。詳しくは『システム管理者の手引き』の「第 8 章メニューの作成」の「コマンド名ファイル」を参照してください。

ユーザコマンドと必要ならば修飾子を下記の書式でコマンド名定義ファイルに追加します。

　+ ［a, b, c］！**コマンド名**！

[a, b, c] は、コマンド識別番号、**！コマンド名！**はコマンド名です。

コマンド名は大文字の英字で始まり、大文字の英字、数字、スラッシュ、またはアンダースコアの文字で構成し、15 文字以内とします。ユーザコマンド名は、システムのコマンド名と重複しないよう U で始めることをお薦めします。
コマンド識別番号は三つの整数からなります。a はディスパッチャ番号、b はドライバ番号、c はフォーム番号と呼びます。

コマンドは下記の表に示すように、ディスパッチャ番号で大きく 8 つに区分されます。そして、区分に対応したコマンドレベルがあります。ただし、コマンドカテゴリ 2 は修飾子であり、実行するコマンドではないのでコマンドレベルはありません。コマンドレベルは数値の大きいものが高い優先度です。優先度の高いコマンドは優先度の低いコマンドを中断して優先実行されます。優先度の高いコマンドが終了すると、以前のレベルのコマンドの実行に戻ります。現在実行中のコマンドと同等かそれ以下のレベルのコマンドが入力されたときは、現在のコマンドは終了させられ、入力コマンドを開始します。

レベル	カテゴリ	ディスパッチャ番号	コマンドの例
1	1	1 ～ 32	**作図、製図、編集**
2	5	81 ～ 88	**トリム、幾何図形の修正**

レベル	カテゴリ	ディスパッチャ番号	コマンドの例
3	4	65 ～ 80	削除、ベリファイ
4	3	49 ～ 64	属性設定
5	6	89 ～ 90	テキストライブラリ
6	7	91 ～ 94	ズーム、再表示
7	8	95 ～ 98	テンポラリポイント
なし	2	33 ～ 48	修飾子

ユーザコマンドを追加できるコマンドレベルは 1、2、3、4 です。修飾子も追加できます。下記の表にユーザコマンドのレベルとディスパッチャ番号を示します。

レベル	カテゴリ	ディスパッチャ番号	種類
1	1	32	コマンド
2	5	88	コマンド
3	4	80	コマンド
4	3	64	コマンド
なし	2	48	修飾子

ユーザコマンド用に以下のコマンド識別番号を確保してあります。

ディスパッチャ番号 ： **上記表を参照**
ドライバ番号 ： 1 ～ 255
フォーム番号 ： 1 ～ 32767

ディスパッチャ番号 32 でフォーム番号が 0 のコマンドはメニュー切り換えをするだけのコマンドです。それ以外の目的には使えません。

2.2 コマンドハンドラの呼び出し

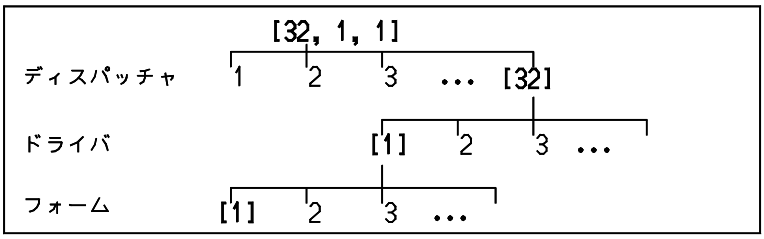
2.2.1 ディスパッチャ関数

最初にアプリケーションがコマンドハンドラを呼び出す手順を説明します。

ユーザがメニュー上のコマンドを選択するか、またはキーボードからコマンド名を入力すると、コマンド名に対応するコマンドの識別番号が得られます。システムはディスパッチャ番号に対応したディスパッチャ関数を呼び出します。関数名は dspatchXX です。XX はディスパッチャ番号です。コマンドレベルごとに用意されたディスパッチャ関数 dspatch32()、dspatch88()、dspatch80()、dspatch64() のどれかが呼び出されます。ここまでは既に実装されています。

ディスパッチャ関数では、ドライバ番号、フォーム番号に対応するコマンドハンドラを呼び出すようにします。通常はコマンドごとにコマンドハンドラを作ります。もし多くのコマンドを追加するならば、ドライバ番号で階層化することができます。たとえば、dspatch32() ではドライバ番号によりドライバ関数 udr01() を呼び出し、そこではさらにフォーム番号によりコマンドハンドラ ucmd01() を呼び出すようにできます。ディスパッチャ関数の名前は決まっていますが、それ以外のドライバ関数、コマンドハンドラ関数の名前は任意です。しかし、システム関数と重複しないようにユーザ関数名は u で始まる名前をつけることをお薦めします。

コマンド識別番号によるコマンドハンドラの呼び出し



ディスパッチャ関数は引数を一つだけ持ちます。それは TOKEN 型のポインタです。TOKEN については後で詳しく説明します。

```
void dspatch32(TOKEN* token) {
}
```

ディスパッチャ関数は最終的に呼び出すコマンドハンドラ関数に引数 TOKEN* token を渡さなければなりません。

```
void ucmd01(TOKEN* token) {
}
```

2.2.2 dspatch32 関数の例

以下はディスパッチャ関数 dspatch32() の例です。コマンド識別番号 [32, 1, 1] は ucmd01() 関数、コマンド識別番号 [32, 1, 2] は ucmd02() 関数、コマンド識別番号 [32, 1, 3] は ucmd03() 関数を呼び出すようにしています。
コマンドカテゴリ 1 のドライバ番号を得るには Idriver(1)、フォーム番号を得るには Iformat(1) を呼出します。Idriver()、Iformat() 関数の引数はコマンドレベルではなく、コマンドカテゴリです。これは歴史的な理由でそうなっています。間違わないようにしてください。

```
void dspatch32(TOKEN* token) {
  switch (Idriver(1)) { // コマンドカテゴリ 1 のドライバ番号
  case 1:
    udr01(token);
    break;
  case 2:
    udr02(token);
    break;
  case 3:
    udr03( token);
    break;
  default:
    break;
  }
} /* dspatch32 */

void udr01(TOKEN* token) {
  switch (Iformat(1)) { // コマンドカテゴリ 1 のフォーム番号
  case 1:
    ucmd01(token);
    break;
  case 2:
    ucmd02(token);
    break;
  case 3:
```

```
      ucmd03(token);
      break;
    default:
      break;
    }
  } /* udr01 */
```

ディスパッチャ関数を実装し呼び出しの径路が決まったら、コマンドの機能を実現するコマンドハンドラを実装します。

2.3 コマンドハンドラ

コマンドハンドラはコマンドの機能を実現する関数です。コマンドハンドラの関数名は任意です。関数は入力引数 TOKEN* token を持たなければなりません。

```
void ucmd01(TOKEN* token) {
}
```

アプリケーションは、コマンドが選択されたとき、ユーザの入力があったとき、コマンドを終了させる時に、その度に先に説明した径路でこのコマンドハンドラを呼び出します。入力は引数 TOKEN* token で渡されます。コマンドハンドラの内容はコマンドの仕様により様々ですが、この関数は 1 つの入力があるたびに呼び出されること、次の入力を得るためにはこの関数を抜け出しアプリケーションの入力処理関数に制御を戻さなければならないことに注意してください。このようなプログラミングは**イベント駆動型プログラミング**と呼ばれます。引数 TOKEN* token でイベントを渡しています。

アプリケーションはコマンドが選択されたときにコマンドハンドラを呼び出します。引数 token のタイプは TknCMD です。このときは、関数の初期設定を行ない、関数を抜け出すようにプログラミングします。次の入力を得るためにはこの関数を抜け出し、システムの入力処理関数に制御を戻さなければならないからです。

アプリケーションは何らかの入力があったら、このコマンドハンドラを呼び出します。入力は引数 token に設定されます。引数 token のタイプは文字列、数値あるいは座標値などで、TknCMD 以外です。TknCMD はコマンドが選択されたときだけです。今度は引数 token のタイプに応じた適切な処理を行ない、次の入力を得るために関数を抜け出すようにプログラミングします。入力一つに対して一回の呼び出しがあります。ここでの処理は、入力がコマンドの構文に適合するかどうか構文解析し、必要なコマンドパラメータが揃った時点で処理を実行するようにプログラミングします。そのため、引数 token で渡されたデータやその個数などを保持しておく必要があります。

アプリケーションは現在のコマンドと同等かそれより低いコマンドレベルのコマンドが選択されると、現在のコマンドを終了させます。コマンドを終了させる前に、システムは終了処理の機会を与えるためこのコマンドハンドラを呼び出します。引数 token のタイプは TknEXIT です。ここでは動的に確保したメモリを開放するなどの後始末をし関数を抜け出すようにプログラミングします。

コマンドハンドラが受け取る一連のトークンの出現順序は次のように表せます。

```
TknCMD、[TknCMD、TknEXIT 以外のトークン]*、TknEXIT
```

関数の骨格は次のようになります。

```
void ucmd01(TOKEN* token) {
  if (token->typ == TknCMD) {
    // 初期化（最初）
  } else if (token->typ == TknEXIT) {
    // 後始末（最後）
  } else {
    // コマンド構文解析、実行
  }
}
```

コマンドハンドラの引数は入力引数なので const TOKEN& token とすべきです。従来通り TOKEN* token としており 引数の内容を変更できますが、変更しないほうが良いでしょう。将来、TOKEN をクラスにし const を付加する可能性があります。また、コマンドハンドラも従来どおり関数で実装しますが、将来はクラスで実装するのが適切だと思われます。その場合は、ディスパッチャ関数は無くなり、コマンドハンドラオブジェクトを生成するファクトリを作成する方法に変わるでしょう。コマンドハンドラオブジェクトがメモリを圧迫する可能性があるので 32bit アプリケーションではなく、64bit アプリケーションにしなければならないでしょう。

2.4 トークン

アプリケーションの入力処理関数はユーザの入力を分析して適切なトークンを生成しコマンドハンドラに渡します。たとえばキーボードから 50,100 と入力された場合は座標トークンに、50 と入力された場合は数値トークンを生成します。
「二点間線」0,0 100,0 <CE> dig1 dig2 「連結線」としたときに「二点間線」コマンドハンドラに渡されるトークンは以下のようになります。

「二点間線」	:	コマンドトークン（TknCMD）
0,0	:	座標トークン（TknCOD）
100,0	:	座標トークン（TknCOD）
<CE>	:	ＣＥ トークン（TknEOC）
dig1	:	デジタイズ座標トークン（TknDIG）
dig2	:	デジタイズ座標トークン（TknDIG）
システムが挿入	:	コマンド終了トークン（TknEXIT）

この後「連結線」コマンドハンドラにコマンドトークンが渡され、「連結線」コマンドが開始します。

トークンは TOKEN 構造体で表現します。TOKEN は acaddef.h に、トークンタイプ定数は acadprm.h にあります。

TOKEN 構造体のメンバー変数 typ がトークンタイプを表します。

TknCMD (== 1) : コマンド
TknMDF (== 16) : 修飾子
TknCOD (== 2) : 座標
TknDIG (== 6) : デジタイズ座標
TknSCL (== 3) : 数値
TknTXT (== 4) : 文字列
TknITM (== 17) : アイテム選択
TknPNT (== 18) : テンポラリポイント
TknIDP (== 15) : アイテム識別子
TknBSP (== 7) : バックスペース
TknSPC (== 11) : スペースキー
TknVEC (== 8) : ベクトル
TknMZN (== 13) : メッセージゾーン
TknGZN (== 12) : グラフィックゾーン
TknUZN (== 14) : テンポラリウインドウ
TknEOC (== 5) : ＣＥ（Enter キー、ボタンに割り付けた<CE>）
TknEXIT (== 31) : コマンド終了

以下に、アプリケーションの入力処理関数がトークンを生成する条件と、値を設定する TOKEN 構造体のメンバー変数を説明します。

● TknCMD : コマンドトークン

コマンドが選択されたときに生成するトークン。
コマンド識別番号を token->cid に設定します。

token->cid[0]	:	ディスパッチャ番号
token->cid[1]	:	ドライバ番号
token->cid[2]	:	フォーム番号

● TknMDF : 修飾子トークン

修飾子が入力されたときに生成するトークン。
修飾子の識別番号を token->cid に設定します。

```
token->cid[0]          :  ディスパッチャ番号
token->cid[1]          :  ドライバ番号
token->cid[2]          :  フォーム番号
```

● TknCOD : 座標トークン（モデル座標）

座標値が入力されたときに生成するトークン。
座標値を token->pnt に設定します。

● TknDIG : デジタイズ座標トークン（モデル座標）

マウスで図形領域をピックしたときに生成するトークン。
ディジタイズが行われたビューポート番号を token->vp に、座標値を token->pnt に設定します。
シフトキー／コントロールキーが押されていたときは token->scl に 0.0 以外を、押されていないときは 0.0 を設定します。

● TknSCL : 数値トークン

数値が入力されたときに生成するトークン。
数値を token->scl に設定します。

● TknTXT : 文字列トークン

文字列が入力されたときに生成するトークン。
文字列を token->txt に設定します。文字列は null-char('\0') で終了します。文字コードは 日本語 EUC（EUC-JP)。

● TknITM : アイテム選択トークン

「複数アイテムの自動選択コマンド」によってアイテムが選択されたときに生成するトークン。
選択されたアイテムはハイライトアイテムリストに格納されている。
詳しくは後述の「例．複数アイテムの自動選択」を参照。

● TknPNT : テンポラリポイントトークン

「テンポラリポイント作成コマンド」によってテンポラリポイントが作成されたときに生成するトークン。テンポラリポイントの座標を token->pnt に設定します。
詳しくは後述の「例．テンポラリポイントの作成」を参照。

● TknIDP : アイテム識別子トークン

マクロの idptr 関数が生成するトークン。通常の操作ではこのトークンは生成されません。アイテム識別子を token->scl に設定します。アイテム識別子は負の整数のときもあります。

● TknBSP : バックスペーストークン

BackSpace キーまたは Del キーが単独で入力されたときに生成するトークン。
文字列や座標値入力中の BackSpace や Del キーの入力は入力処理関数が処理を行うのでこのトークンは発生しません。

● TknSPC : スペースキートークン

スペースキーが単独で入力されたときに生成するトークン。

● TknVEC : ベクトルトークン

@DX10 などのベクトル入力や割り込みのベクトルコマンドでベクトルが作成されときに生成するトークン。
ベクトル値 DX、DY を token->pnt に設定します。

● TknMZN : メッセージゾーントークン

メッセージゾーンがピックされたときに生成するトークン。
ピックされたメッセージゾーンの位置（行、列）を token->pnt に設定します。

```
token->pnt.x          :  列番号 (1-)
token->pnt.y          :  行番号 (1-)
```

● TknGZN : グラフィックゾーントークン

グラフィックゾーンがピックされたときに生成するトークン（tknmsk001() 関数で入力可にしたした場合に限る)。
ピックされたグラフィックゾーンの位置（行、列）を token->pnt に設定します。

```
token->pnt.x          :  列番号 (1-)
token->pnt.y          :  行番号 (1-)
```

● TknUZN : テンポラリウインドウトークン

マクロの mopen () 関数によって作成されたテンポラリウインドウがピックされたときに生成するトークン（tknmsk001() 関数で入力可にした場合に限る)。
ピックされたテンポラリウインドウの位置（行、列）を token->pnt に設定します。

```
token->pnt.x          :  列番号 (1-)
token->pnt.y          :  行番号 (1-)
```

● TknEOC : ＣＥ トークン

Enter キーまたはボタンに割り付けた <CE> が入力されたときに生成するトークン。

● TknEXIT : コマンド終了トークン

コマンドが切り替るときに生成するトークン。
現在のコマンドと同等かそれより低いコマンドレベルのコマンドに切り替わるときに生成する。
F1 キー入力（コマンドの強制終了）のときもこのトークンを生成する。

2.5 コマンドの自発的終了

コマンドレベル 2 以上のコマンドは、自分よりコマンドレベルの低いコマンドの実行を中断し優先実行するので、割り込みコマンドと呼ぶことがあります。割り込みコマンドのハンドラは自発的に終了できるようにプログラミングします。そうでないと、コマンドレベルが自分と同等かそれより低いコマンドが入力されないと終了できないことになります。
自発的終了するタイミングはコマンド構文に依存します。割り込みコマンドのコマンド構文は <CE> で終了する仕様が多いので、CE トークンが渡されたときに終了処理をする例を示します。自発的終了は cmdidcla() 関数で行います。この関数の引数はコマンドカテゴリです。

```
if (token->typ == TknEOC) {
  // ここで後始末をする。
  cmdidcla(1);
  return;
}
```

自発的終了した場合は TknEXIT トークンでの呼び出しはなされませんので、後始末をしてから終了します。それ以外の通常の終了は TknEXIT トークンでの呼び出しが行われます。

2.6 修飾子

修飾子は単独で使用することもありますが、引数を伴う使い方もあります。例えばコマンド構文に角度を入力する ANG s があるとします。これは、修飾子トークン、数値トークンの順に別々にコマンドハンドラに渡されます。また、ユーザが誤って数値でなく、文字列を入力するかもしれません。ここでは

引数を伴う修飾子を処理する方法を説明します。例として ANG s を使います。修飾子 ANG の識別番号は [34, 1, 4] です。
修飾子トークンが渡されたら修飾子 ANG か確認します。修飾子の識別番号は TOKEN 構造体のメンバー変数 cid にあります。修飾子 ANG を確認したならば「文字角度を入力」などのメッセージを表示します。期待した修飾子でないときは、cmdmdfcla() 関数で修飾子をクリアします。この関数の引数はコマンドカテゴリです。

```
if (token->cid[0] == 34 && token->cid[1] == 1 && token->cid[2] == CSWANG) {
  // ここで角度入力を促すメッセージを表示する。
} else {
  cmdmdfcla(1);
  // 期待した修飾子でない。ここでエラーメッセージを表示する。
}
```

数値トークンが渡されたら、先に修飾子 ANG が入力されていることを確認します。先に入力された修飾子のコマンド番号を得るには、ldispatch()、ldriver()、lformat() 関数を使います。関数の引数は修飾子のコマンドカテゴリ 2 です。修飾子 ANG を確認したならば、TOKEN 構造体のメンバー変数 scl が角度です。これで修飾子 ANG の処理が完了したので修飾子をクリアしておきます。

```
if (ldispatch(2) == 34 && ldriver(2) == 1 && lformat(2) == CSWANG) {
  angle = token->scl;
  cmdmdfcla(1);
  // ここで角度を表示する。
}
```

ANG s の処理をするコマンドハンドラは次のようになります。

```
void ucmd01(TOKEN* token) {
  static double angle;
  if (token->typ == TknCMD) {
    angle = 0.0; // static 変数を初期化。前回の値をそのまま使うなら何もしない。
  } else if (token->typ == TknMDF) {
    if (token->cid[0] == 34 && token->cid[1] == 1 && token->cid[2] == CSWANG) {
      // ここで角度入力を促すメッセージを表示する。
    } else {
      cmdmdfcla(1);
      // 期待した修飾子でない。ここでエラーメッセージを表示する。
    }
  } else if (token->typ == TknSCL) {
    if (ldispatch(2) == 34 && ldriver(2) == 1 && lformat(2) == CSWANG) {
      angle = token->scl; //  static 変数に代入。
      cmdmdfcla(1);
      // ここで角度を表示する。
    }
  } else if (token->typ == TknEXIT) {
    // 後始末だが何もしない。
  } else {
    // 期待したトークンでない。
  }
}
```

2.7 トークンマスクを使う

入力可能なトークンの種類を設定することができます。これはデフォルトでは生成されないトークンタイプを得たいときに必要です。たとえば、グラフィックゾーンをディジタイズしたときは、モデル座標を持つ TknDIG トークンを生成します。モデル座標ではなく、ゾーンの位置（行、列）を持つ TknGZN トークンを得るには、あらかじめ TknGZN トークンを入力可能にする必要があります。そうした場合は TknDIG トークンは入力できなくなります。
また、入力可能なトークンの種類を限定する時にも使います。

入力可能なトークン種類を設定するには tknmsk001() 関数を使います。コマンドハンドラがこの関数で行った設定は、設定直後の入力一回だけに有効で、トークンが生成された後にクリアされ、デフォルトに戻ります。

2.8 テンポラリポイントモードの適用

アプリケーション にはテンポラリポイントモードがあります。テンポラリポイントモードは「自動」「端点」[中点」「中心点」「ディジタイズ点」「交点」「投影点」などがあり、それらのどれかひとつが選択されています。
テンポラリポイントモードの適用はコマンドハンドラで行います。TknDIG トークンはグラフィックゾーンのディジタイズした位置に対応するモデル座標を持っています。このモデル座標をもとにテンポラリポイントモードを適用した点を得るには IdentPoint() 関数を使います。この関数は第 1 引数のトークンを入力とし、計算した点を第 2 引数に設定します。関数は成功すると IDENT_SUCCESS を返します。

```
if (token->typ == TknDIG) {
  DPOINT pnt;
  if (IdentPoint(token, &pnt) != IDENT_SUCCESS)
    return;
}
```

テンポラリポイントモードの「交点」、「投影点」などは複数の入力が必要です。たとえば、「交点」は 2 つのモデル座標が必要ですが、コマンドハンドラは IdentPoint() 関数に最初のモデル座標しか渡せません。このような場合、関数は最初のモデル座標の処理が成功したら IDENT_CONTINUE を返します。これは、関数が「交点」を処理するために高優先度の「テンポラリポイントコマンド」を起動したことを意味します。「交点」の処理はテンポラリポイントコマンドが引き継ぎます。テンポラリポイントコマンドは成功すると、計算した点を持つ TknPNT トークンを生成します。次の例はこの処理を追加したものです。

```
DPOINT pnt;
if (token->typ == TknDIG) {
  if (IdentPoint(token, &pnt) != IDENT_SUCCESS)
    return; // テンポラリポイントコマンドを起動した。あるいはエラー。
} else if (token->typ == TknPNT) {
  pnt = token->pnt; // テンポラリポイントコマンドの結果。
}
```

IdentPoint() 関数は TknPNT トークンもうまく処理するので , 次のようにまとめて記述することができます。

```
if (token->typ == TknDIG || token->typ == TknPNT) {
  DPOINT pnt;
  if (IdentPoint(token, &pnt) != IDENT_SUCCESS)
    return;
}
```

2.9 複数のアイテムの選択

アイテムを選択するには IdentItems() 関数を使います。関数の第 1 引数はコマンドカテゴリ、第 2 引数に TknDIG または TknCOD トークンを渡します。関数はアイテム選択が成功すると IDENT_SUCCESS を返します。

```
if (token->typ == TknDIG || token->typ == TknCOD) {
  if (IdentItems(1, token, 0) != IDENT_SUCCESS)
    return;
}
```

関数はアイテム選択ができなかったら IDENT_CONTINUE を返します。これは、関数に渡された点の近傍にはアイテムが無いので、「矩形」または「多角形」領域内のアイテム選択に切り替えるため、高優

先度の「アイテム選択コマンド」を起動したことを意味します。関数に渡した点はアイテム選択コマンドが引き継ぎます。アイテム選択コマンドは終了すると TknITM トークンを生成します。TknITM トークンを関数に渡してアイテム選択コマンドの結果を判定します。次の例はこの処理を追加したものです。

```
if (token->typ == TknDIG || token->typ == TknCOD) {
  if (IdentItems(1, token) != IDENT_SUCCESS)
    return; // アイテム選択コマンドを起動した。
} else if (token->typ == TknITM) {
  if (IdentItems(1, token) != IDENT_SUCCESS)
    return; // アイテム選択コマンドの結果判定。
}
```

これは次のようにまとめて記述することができます。

```
if (token->typ == TknDIG || token->typ == TknCOD || token->typ == TknITM) {
  if (IdentItems(1, token) != IDENT_SUCCESS)
    return;
}
```

IdentItems() 関数は選択したアイテムをハイライトアイテムリストに追加します。

2.10 アイテム選択の詳細情報

アイテムを選択するには IdentItem() 関数を使います。関数の第 1 引数はコマンドカテゴリ、第 2 引数に TknDIG、TknCOD、TknIDP または TknSCL トークンを渡します。関数は選択したアイテムのアイテム識別子を返します。アイテム識別子は正の整数です。失敗したときは 0 を返します (IdentItems() 関数とは違います)。

```
int itemid = IdentItem(1, token, 0);
if (itemid <= 0)
    return;
```

IdentItem() 関数は TknDIG、TknCOD トークンを受け取ったときは、トークンの点 (token->pnt) の近傍にある複数のアイテムを候補として選びます。そして第 1 候補（点に最も近い所にあるアイテム）を返します。いくつの候補があったかを知るには IdentItemCandidate() 関数を使います。引数はコマンドカテゴリです。候補数が 2 以上のときは次の候補を取り出す簡単な方法があります。IdentItem() 関数に TknSPC を渡せば次の候補が得られます。これを繰り返すと順番に候補をたどり巡回します。

アイテム選択の詳細情報を知りたければ IdentInfo() 関数を使います。この関数は IDENTINFO 構造体へのポインタを返します。

```
const IDENTINFO* info = IdentInfo();
```

この情報は IdentItem() 関数が設定したものです。IdentItem() 関数が失敗したときはこのポインタは null です。実際には info は IDENTINFO の配列で、配列サイズが候補数と思ってください。
info->idptr と info[0]->idptr は同じです。

IDENTINFO 構造体は acaddef.h にあり、以下のメンバー変数を持っています。

int idptr	アイテム識別子 (1 ー)。
int snum	ピックしたサブレコードのインデックス（0 ー)。
int styp	ピックしたサブレコードの種類。
int wend	幾何図形要素のどちらの端点に近い方を指示したか。 1 = 始点、-1 = 終点。ピックしたサブレコードの種類が曲線（線分・円・Bezier 曲線）のときのみ有効です。それ以外は 1 です。
double dist	指示した位置と曲線上に投影した点との距離。
DPOINT ploc	指示した位置をピックした曲線上に投影した点。

DGEOM geom　　サブレコードから取り出したデータ。

サブレコード	データ
AwSr::POINT	geom.pnt
AwSr::LINE	geom.lin
AwSr::CIRCLE	geom.arc
AwSr::BEZIER	geom.bzc
AwSr::TEXT	Po, P1, P2, P3, Px（Px は文字列の位置）
AwSr::MARK	Po, P1, P2, P3, Px（Px はノードの位置）

int ctgsridx　　ピックしたサブレコードの前に現れた分類サブレコード（AwSr::CATEGORY）のインデックス（-1 : なし , 0 ー）。

short pid　　図面配置状態のとき、ピックされたアイテムの属するピクチャ番号。(1 -) 0 = 図面配置状態ではない。

short wid　　図面配置状態のとき、ピックされたアイテムの属するウインドウ番号。

2.11 ハイライトリスト

コマンドハンドラでは、ある条件が揃うまで、選択したアイテムの識別子や入力した点列を記憶しておくことがよくあります。そして選択したアイテムの識別子や入力した点列が明確にわかるように画面に表示しておくことが望ましいのです。このような目的で、コマンドハンドラに可変長の記憶領域を提供するのがハイライトリストです。
ハイライトリストのデータはスクリーンのテンポラリ・プレーンに表示します。通常、テンポラリ・プレーンは白色表示です。スクリーンをズーム、パン、再表示などしてテンポラリ・プレーンが再描画されるときにはハイライトリスト内のデータも再表示されます。

ハイライトリストは割り込みコマンドを考慮してコマンドのレベルごとに用意しています。コマンドハンドラが終了したとき、ハイライトリストはクリアしてデフォルトに戻されます。前のコマンドが使った結果が残っていることはありません。
ハイライトリストを得るには GetHighlightSet() 関数を使います。関数の引数はコマンドカテゴリです。
ハイライトリストで良く使われるのは「アイテムリスト」と「点列リスト」です。以下のコードフラグメントはそれらの使い方の概略を示します。

アイテムリストの例

```
AwHighlightItems* pHlItems = GetHighlightSet(1)->GetHlItems();
if (token->typ == TknCMD) {
  // この時点ではリストは初期化されている。
  // 表示方法や最大アイテム数を変更したければここで行う。
  // pHlItems->SetMaxCount(1024);
} else if (token->typ == TknCOD || token->typ == TknDIG) {
  int id = IdentItem(1, token);
  if (pHlItems->Append(1,&id)  // リストにアイテム識別子を追加する。
    pHlItems->DrawLast();    // 追加したアイテムを表示する。
} else if (token->typ == TknBSP) {
  // リストから最後のアイテム識別子を除去する。
  if (pHlItems->RemoveLast()) {
    Vieerase(0, TEMPSCREEN); // テンポラリ・プレーンをイレース
    rpttmppln();             // テンポラリ・プレーンを再描画
  }
} else if (token->typ == TknEOC) {
  const AwItemIdArray& itemids = pHlItems->GetItemIds();
  // ここで itemids を使って何か処理をする。
```

```
    pHlItems->RemoveAll();      // リストをクリアする
    Vieerase(0, TEMPSCREEN);    // テンポラリ・プレーンをイレース
  }
```

点列リストの例

```
  AwHighlightPoints* pHlPoints = GetHighlightSet(1)->GetHlPoints();
  if (token->typ == TknCMD) {
    // この時点ではリストは初期化されている。
    // 表示方法や最大点数を変更したければここで行う。
    // pHlPoints->SetMaxCount(16);
  } else if (token->typ == TknCOD || token->typ == TknDIG) {
    if (pHlPoints->Add(token->pnt))
      pHlPoints->DrawLast();   // 追加した点を表示する。
  } else if (token->typ == TknBSP) {
    // リストから最後の点を除去する。
    if (pHlPoints->RemoveLast()) {
      Vieerase(0, TEMPSCREEN); // テンポラリ・プレーンをイレース
      rpttmppln();             // テンポラリ・プレーンを再描画
    }
  } else if (token->typ == TknEOC) {
    const G2PointArray& points = pHlPoints->GetPoints();
    // ここで points を使って何か処理をする。
    pHlPoints->RemoveAll();     // リストをクリアする
    Vieerase(0, TEMPSCREEN);    // テンポラリ・プレーンをイレース
  }
```

この例は分かりやすさのため「アイテムリスト」と「点列リスト」を別々に示しましたが、この二つをを同時に使用しても問題ありません。
また、IdentItems() 関数は選択したアイテムをここで説明した「アイテムリスト」に格納することを思い出してください。IdentItems() 関数を使用しながら「アイテムリスト」を直接操作すると混乱するかもしれません。

2.12 ドラッギング

ドラッギングは、あらかじめ図形を登録しておき、それをドラッグします。
ドラッグする図形を登録するとき、アイテムは線分、円弧、Bezier 曲線セグメントに分解して登録します。文字やマークも同様に分解して登録します。ドラッグする図形を保持する領域の大きさには制限があり、その制限を超えた分は保持できません。ドラッギングはマウスの移動にあわせて描画、消去を高速に行う必要があります。この容量を大きくするのは、ドラッギングの追従性が低下するので、効果的ではありません。
ドラッギングは割り込みコマンドを考慮してコマンドのレベルごとに用意しています。コマンドハンドラが終了したとき、ドラッグする図形はクリアされます。前のコマンドが使った結果が残っていることはありません。

最初に Dragopen () 関数でドラッギングの種類を指示します。Dragopen () 関数は AwDragLoader オブジェクトへのポインタを返します。続いて AwDragLoader オブジェクトを使ってドラッグする図形を登録します。
アプリケーションの入力処理関数に制御が戻るとドラッギングが開始します。
ドラッギングを終了させるには、Dragend() 関数を使います。

次のコードフラグメントはピックしたアイテムをドラッグする図形として登録する例です。

```
  const AwItem* pItem = PickItem(1, *token, 0);
  if (pItem == NULL)
    return;
  const DPOINT pnts[2] = { {0.0, 0.0}, {0.0, 0.0} };
  AwDragLoader* pDragLoader = Dragopen(1, 1, pnts); // 初期化する
  if (pDragLoader)
```

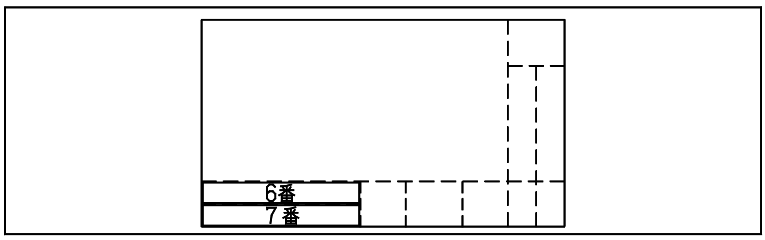
```
pDragLoader->LoadItem(*pItem);  // ドッラッギングデータを追加する
```

2.13 メッセージ出力

コマンドハンドラは、メッセージを、グラフィックスクリーンの一部に確保したメッセージ用ゾーンに出力します。

メッセージ用ゾーンは、６番と７番の２つです。６番のゾーンには、コマンドの状態を表示します。例えばユーザの入力や、計算結果を表示します。７番のゾーンは、オペレータにつぎの操作を促すオペレーションメッセージとエラーメッセージを表示します。

標準メニューを使用時のメッセージゾーン 6番、7番



オペレーションメッセージや、エラーメッセージはすべてメッセージファイルに登録します。ユーザ用のメッセージファイルは、MSG90.TXT (オペレーションメッセージファイル) と ERR90.TXT (エラーメッセージファイル) の２つです。詳しくは『システム管理者の手引き』の「メッセージの修正」の章を参照してください。

メッセージは次の書式で１行で記述します。

+（メッセージ番号）″メッセージ″

メッセージおよびエラーメッセージ番号は、それぞれ 9000000 から 9999999 です。
メッセージとエラーメッセージは独立しているので同じ番号でもかまいません。

メッセージ番号を重複して指定すると、最初に現われたものが有効になります。
このため、MSG.INP 内で !MSG90! を ERR.INP 内で !ERR90! を一番最初に記述してあります。

以下の関数は、メッセージ番号を指定するとメッセージを所定の位置に表示します。

Opmsgcode	オペレーションメッセージ表示
Errorcode	エラーメッセージ表示
Errorb	ブザーを鳴らす
Mesagdisp	メッセージ表示
Mesageras	メッセージの消去

2.14 例

2.14.1 座標値、数値および文字列トークンの取り扱い

```
/*
 * 入力された座標値、数値および文字列をメッセージ領域に表示する。
 * <CE> でこのコマンドを終了する。
 */
```

```
#include "acaddef.h"
#include "acadprm.h"
#include "acadusr.h"
#include "acadupi.h"

#define CMDCTG 3 /* コマンドカテゴリ３の割り込みコマンド */

void ucmd01(TOKEN* token) {
  switch (token->typ) {

  case TknCMD: /* コマンドトークン */
    break;

  case TknCOD: /* 座標トークン */
  case TknDIG: /* デジタイズ座標トークン */
    /*
     * 入力された座標値をメッセージ領域に表示する。
     * メッセージ番号 9000001 : "座標値 X"
     * メッセージ番号 9000002 : "座標値 Y"
     */
    Mesageras(MSGZONE, 0, 0);
    Mesagdisp(MZONECOLOR1, 1, 1, 9000001, 3, &token->pnt.x);
    Mesagdisp(MZONECOLOR1, 2, 1, 9000002, 3, &token->pnt.y);
    break;

  case TknSCL: /* 数値トークン */
    /*
     * 入力された数値をメッセージ領域に表示する。
     * メッセージ番号 9000003 : "数値"
     */
    Mesageras(MSGZONE, 0, 0);
    Mesagdisp(MZONECOLOR1, 1, 1, 9000003, 3, &token->scl);
    break;

  case TknTXT: /* 文字列トークン */
    /*
     * 入力された文字列をメッセージ領域に表示する。
     * メッセージ番号 9000004 : "文字列"
     */
    Mesageras(MSGZONE, 0, 0);
    Mesagdisp(MZONECOLOR1, 1, 1,9000004, strlen(token->txt) * 10, token->txt);
    break;

  case TknEOC: /* ＣＥ トークン */
    /*
     * Advance CAD では ＣＥ トークンの場合は概ね以下のように処理している。
     * 基本コマンドの場合は今までの操作で設定された条件に基づき処理を行う。
     * その後このコマンド処理関数を初期状態に戻して次のトークンの入力を待つ。
     * 割り込みコマンドの場合はコマンド処理を終了させる。
     * この例ではコマンドカテゴリ３の割り込みコマンドであるので cmdidcla 関数
     * を呼出してこのコマンドを終了する。
     * これ以後に入力されたトークンは現在動作中のより下位の割り込みコマンド
     * または基本コマンドに渡されるようになる。
     */
    cmdidcla(CMDCTG);
    return;

  case TknEXIT: /* コマンド終了トークン */
    return;

  default: /* その他のトークン */
```

```
    Errorcode(9000001); /* 無効な入力です */
    break;
  }

  /* 入力促進メッセージ */
  Opmsgcode(CMDCTG, 9000005); /* 座標値、数値、文字列を入力／終了は<CE> */
}
```

2.14.2 修飾子トークンの取り扱い

```
/*
 * 修飾子により、入力された数値が文字高さか文字角度かを判定する。
 */

#include "acaddef.h"
#include "acadprm.h"
#include "acadusr.h"
#include "acadupi.h"

#define CMDCTG 1 /* 基本コマンド */

void ucmd01(TOKEN* token) {
  switch (token->typ) {

  case TknCMD: /* コマンドトークン */
    break;

  case TknMDF: /* 修飾子トークン */
    if (token->cid[0] == 34 &&
        token->cid[1] == 1 &&
        token->cid[2] == CSWTSIZE) { /* 修飾子 TSIZE */
      /* EMPTY */
    } else if (token->cid[0] == 34 &&
               token->cid[1] == 1 &&
               token->cid[2] == CSWANG) { /* 修飾子 ANG */
      /* EMPTY */
    } else { /* その他の修飾子 */
      Errorcode(9000001); /* 無効な修飾子です */
      /* 無効な修飾子をクリアする。*/
      cmdmdfcla(CMDCTG);
    }
    break;

  case TknSCL: /* 数値トークン */
    if (ldispatch(2) == 34 &&
        ldriver(2) == 1 &&
        lformat(2) == CSWTSIZE) {
      /*
       * 修飾子 TSIZE が指定されている。
       * 入力された数値は文字高さ。文字高さをメッセージ領域に表示する。
       * メッセージ番号 9000001 : "文字高さ"
       */
      Mesageras(MSGZONE, 1, 0);
      Mesagdisp(MZONECOLOR1, 1, 1, 9000001, 3, &token->scl);
    } else if (ldispatch(2) == 34 &&
               ldriver(2) == 1 &&
               lformat(2) == CSWANG) {
      /*
       * 修飾子 ANG が指定されている。
       * 入力された数値は文字角度。文字角度をメッセージ領域に表示する。
```

```
       * メッセージ番号 9000002 : " 文字角度 "
       */
      Mesageras(MSGZONE, 2, 0);
      Mesagdisp(MZONECOLOR1, 2, 1, 9000002, 3, &token->scl);
    } else {
      Errorcode(9000002); /* 修飾子が指定されていない */
    }
    break;

  case TknEOC: /* ＣＥ トークン */
    break;

  case TknEXIT: /* コマンド終了トークン */
    return;

  default: /* その他のトークン */
    Errorcode(9000003); /* 無効な入力です */
    break;
  }

  /* 入力促進メッセージ */
  if (Idispatch(2) == 34 &&
      Idriver(2) == 1 &&
      Iformat(2) == CSWTSIZE) {
    /* 修飾子 TSIZE が指定されている。*/
    Opmsgcode(CMDCTG, 9000003); /* 文字高さを入力 */
  } else if (Idispatch(2) == 34 &&
             Idriver(2) == 1 &&
             Iformat(2) == CSWANG) {
    /* 修飾子 ANG が指定されている。*/
    Opmsgcode(CMDCTG, 9000004); /* 文字角度を入力 */
  } else {
    /* 修飾子が指定されていない。*/
    Opmsgcode(CMDCTG, 9000005); /* 修飾子「文字高さ」「文字角度」を選択 */
  }
}
```

2.14.3 テンポラリポイントの作成

```
/*
 * 入力された二点間の距離をメッセージ領域に表示する。
 */

#include "acaddef.h"
#include "acadprm.h"
#include "acadusr.h"
#include "acadupi.h"

#define CMDCTG 1 /* 基本コマンド */

void ucmd01(TOKEN* token) {
  static G2Point pnts[2]; /* テンポラリポイントの保存領域 */
  static int npnt = 0;   /* 入力済みのテンポラリポイントの点数 */

  switch (token->typ) {

  case TknCMD: /* コマンドトークン */
    /*
     * 初期設定。
     */
```

```
  npnt = 0;
  break;

case TknCOD: /* 座標トークン */
case TknDIG: /* デジタイズ座標トークン */
case TknPNT: /* テンポラリポイントトークン */
  /*
   * テンポラリポイントを作成する。
   * IdentPoint 関数の概要
   * TknCOD が渡されたときの戻り値
   *   IDENT_SUCCESS  : 渡された点をテンポラリポイントとする。
   * TknDIG が渡されたときの戻り値
   *   IDENT_SUCCESS  : テンポラリポイントが作成できた。
   *   IDENT_FAILURE  : テンポラリポイントが作成できない。
   *   IDENT_CONTINUE : テンポラリポイントモードが交点や投影点などであり、
   *                    指定された１点ではテンポラリポイントが作成できな
   *                    いのでテンポラリポイント作成コマンド（割り込みコ
   *                    マンド）を起動した。
   *                    テンポラリポイント作成コマンドが終了すると、
   *                    その結果はトークンタイプ TknPNT で通知される。
   * TknPNT が渡されたときの戻り値
   *   IDENT_SUCCESS  : テンポラリポイントが作成できた。
   *   IDENT_FAILURE  : テンポラリポイントが作成できない。
   * 戻り値が IDENT_SUCCESS のときは２番目の引数にテンポラリポイントが設定されている。
   * 戻り値が IDENT_FAILURE のときは IdentPoint 関数内でエラーメッセージを表示している。
   */
  if (PickPoint(token, &pnts[npnt]) != IDENT_SUCCESS)
    return;
  /* テンポラリポイントが作成できた。*/
  ++npnt;
  if (npnt == 2) {
      /*
       * 二点間の距離を計算しメッセージ領域に表示する。
       * メッセージ番号 9000001 : "距離"
       */
      double dist = pnts[0].CalcDistance(pnts[1]);
      Mesageras(MSGZONE, 0, 0);
      Mesagdisp(MZONECOLOR1, 1, 1, 9000001, 3, &dist);
      npnt = 0;
  }
  break;

case TknBSP: /* バックスペーストークン */
  if (npnt == 1) {
    npnt = 0; /* １点目のテンポラリポイントをキャンセルする。*/
  } else {
    Errorcode(9000001); /* 取り消すテンポラリポイントがない */
  }
  break;

case TknEOC: /* ＣＥ トークン */
  npnt = 0; /* 初期化する。*/
  break;

case TknEXIT: /* コマンド終了トークン */
  return;

default: /* その他のトークン */
  Errorcode(9000002); /* 無効な入力です */
  break;
}
```

```
  /* 入力促進メッセージ */
  if (npnt == 0) {
    Opmsgcode(CMDCTG, 9000002); /* 線分の１点目を入力 */
  } else {
    Opmsgcode(CMDCTG, 9000003); /* 線分の２点目を入力 */
  }
}
```

2.14.4 アイテムの選択

```
/*
 * 選択された曲線アイテムの中点をメッセージ領域に表示する。
 */

#include "acaddef.h"
#include "acadprm.h"
#include "acadusr.h"
#include "acadupi.h"

#define CMDCTG 1

void ucmd01(TOKEN* token) {

  switch (token->typ) {

  case TknCMD: /* コマンドトークン */
    /*
     * 初期設定。
     * IdentItem 関数の初期化を行う。
     * 入力処理関数は、トークンタイプ TknCMD でコマンド処理関数を呼出す前に
     * IdentItem 関数を初期化している。従って下の１行は不要。あってもよい。
     */
    //(void)IdentItem(CMDCTG);
    break;

  case TknCOD: /* 座標トークン */
  case TknDIG: /* デジタイズ座標トークン */
  case TknIDP: /* アイテム識別子トークン */
  case TknSPC: /* スペースキートークン */
    /*
     * アイテムを選択する。
     * IdentItem 関数の概要
     * TknCOD、TknDIG が渡されたときの処理
     *   渡された点でアイテムをピックする。
     *   ピックできればそのアイテムのアイテム識別子を返す。
     *   ピックできなければ 0 を返す。
     * TknIDP が渡されたときの処理
     *   渡されたアイテム識別子を調べる。
     *   アイテム識別子が正しければそのアイテム識別子を返す。
     *   アイテム識別子が正しくなければ 0 を返す。
     * TknSPC が渡されたときの処理
     *   次候補アイテムの有無を調べる。
     *   次候補アイテムがあればそのアイテムのアイテム識別子を返す。
     *   次候補アイテムがなければ 0 を返す。
     */
    {
      const AwItem* pItem = PickItem(CMDCTG, *token, 0);
      if (pItem == 0) {
        /* アイテムが選択できなかった */
        Errorcode(9000001);
```

```
      } else {
        /*
         * アイテムが選択できた。
         * 選択された曲線アイテムの中点座標をメッセージ領域に表示する。
         * メッセージ番号 9000001 : "中点 X"
         * メッセージ番号 9000002 : "中点 Y"
         */
        Mesageras(MSGZONE, 0, 0);
        G2PointArray pointArray;
        AwCurveOperator curveOperator(*(GetActiveModel()->GetGeomParam()));
        if (curveOperator.CalcPoints(*pItem, maxpnt, &pointArray)) {
          const G2Point& pmid = pointArray.GetAt(0);
          Mesagdisp(MZONECOLOR1, 1, 1, 9000001, 3, &pmid.x);
          Mesagdisp(MZONECOLOR1, 2, 1, 9000002, 3, &pmid.y);
        }
      }
    }
    break;

  case TknEOC: /* ＣＥ トークン */
    break;

  case TknEXIT: /* コマンド終了トークン */
    return;

  default: /* その他のトークン */
    Errorcode(9000002); /* 無効な入力です */
    break;
  }

  /* 入力促進メッセージ */
  if (IdentItemCandidate(CMDCTG) >= 2) { /* 次候補アイテムの有無を調べる */
    /* 次候補アイテムあり。*/
    Opmsgcode(CMDCTG, 9000003); /* 中点を求める曲線アイテムを選択または<SP>（次候補）を入力 */
  } else {
    /* 次候補アイテムなし。*/
    Opmsgcode(CMDCTG, 9000004); /* 中点を求める曲線アイテムを選択 */
  }
}
```

2.14.5 複数アイテムの自動選択

※V19 からは次候補アイテムの有無を調べる方法が変更になりました。

```
/*
 * 複数アイテムを選択し、現在選択されているアイテム数と最後に選択された
 * アイテムのアイテム識別子をメッセージ領域に表示する。
 * <CE> で選択状態を初期化する。
 */

#include "acaddef.h"
#include "acadprm.h"
#include "acadusr.h"
#include "acadupi.h"

#define CMDCTG 1

void ucmd01(TOKEN* token) {
```

```
switch (token->typ) {

case TknCMD: /* コマンドトークン */
  /*
   * 初期設定。
   * IdentItems 関数の初期化を行う。
   * 入力処理関数は、トークンタイプ TknCMD でコマンド処理関数を呼出す前に
   * IdentItems 関数を初期化している。従って下の１行は不要。あってもよい。
   */
  //(void)IdentItems(CMDCTG);
  break;

case TknCOD: /* 座標トークン */
case TknDIG: /* デジタイズ座標トークン */
case TknIDP: /* アイテム識別子トークン */
case TknSPC: /* スペースキートークン */
case TknBSP: /* バックスペーストークン */
case TknITM: /* アイテム選択トークン */
  /*
   * アイテムを選択する。
   * IdentItems 関数の概要
   * TknCOD、TknDIG が渡されたときの戻り値
   *   IDENT_SUCCESS  : アイテムがピックできた。
   *   IDENT_CONTINUE : 渡された点でアイテムがピックできなかったので
   *                    矩形または多角形領域でのアイテム選択とみなし、
   *                    複数アイテムの自動選択コマンド（割り込み
   *                    コマンド）を起動した。
   *                    複数アイテムの自動選択コマンドが終了すると、
   *                    その結果はトークンタイプ TknITM で通知される。
   * TknIDP が渡されたときの戻り値
   *   IDENT_SUCCESS  : アイテムが選択できた。
   *   IDENT_FAILURE  : アイテムが選択できなかった。（渡されたアイテム識別子が不正）
   * TknSPC が渡されたときの戻り値
   *   IDENT_SUCCESS  : 次候補アイテムあり。
   *                    以前のアイテムを排除し、次候補アイテムを選択する。
   *   IDENT_FAILURE  : 次候補アイテムはない。
   * TknBSP が渡されたときの戻り値
   *   IDENT_SUCCESS  : 最後の操作を元に戻した。
   *   IDENT_FAILURE  : 元に戻すべきアイテムが存在しない。
   * TknITM が渡されたときの戻り値
   *   IDENT_SUCCESS  : 複数アイテムの自動選択コマンドでアイテムが選択できた。
   *   IDENT_FAILURE  : アイテムが選択できなかった。
   * 戻り値が IDENT_SUCCESS のときは、選択されたアイテムはハイライト
   * アイテムリストに追加されている（または排除されている）。
   * 戻り値が IDENT_FAILURE のときは IdentItems 関数内でエラーメッセージを表示している。
   */
  if (IdentItems(CMDCTG, token, 0) != IDENT_SUCCESS) {
    return;
  } else {
    /*
     * アイテムが選択／排除できた。
     * 選択されているアイテム数および最後に選択されたアイテムの
     *アイテム識別子をメッセージ領域に表示する。
     * メッセージ番号 9000001 : "アイテム数"
     * メッセージ番号 9000002 : "アイテム識別子"
     */
    const AwHighlightItems* pHlItems = GetHighlightSet(CMDCTG)->GetHlItems();
    int nitem = pHlItems->GetCount(); /* ハイライトアイテムリスト内のアイテム数 */
    Mesageras(MSGZONE, 0, 0);
    Mesagdisp(MZONECOLOR1, 1, 1, 9000001, 4, &nitem);
    if (nitem > 0) {
      int idptr = pHlItems->GetLast(); /* 最後に選択されたアイテムのアイテム識別子 */
```

```
            Mesagdisp(MZONECOLOR1, 2, 1, 9000002, 4, &idptr);
          }
        }
        break;

      case TknEOC: /* ＣＥ トークン */
        /* 選択状態を初期化する（Identltems 関数の初期化を行う）。*/
        (void)Identltems(CMDCTG);
        break;

      case TknEXIT: /* コマンド終了トークン */
        return;

      default: /* その他のトークン */
        Errorcode(9000001); /* 無効な入力です */
        break;
      }

      /* 入力促進メッセージ */
      if (IdentltemsCandidate(CMDCTG) >= 2) { /* 次候補アイテムの有無を調べる */
        /* 次候補アイテムあり。*/
        Opmsgcode(CMDCTG, 9000003); /* アイテムを選択または<SP>（次候補）を入力 */
      } else {
        /* 次候補アイテムなし。*/
        Opmsgcode(CMDCTG, 9000004); /* アイテムを選択 */
      }
    }
```

第 3 章　アイテム操作の基本

この章ではアイテムを作成したり、アイテムのデータを参照する方法を説明します。最初に、モデルの構造を説明します。モデルとモデルを構成している要素は C++ のクラスで実装されています。ここでは主なクラスの概要を説明するにとどめます。それらの詳細は後の章で説明します。具体的にプログラミングの例を示しながらアイテムのデータ参照方法やアイテムの作成方法などを説明します。

3.1　モデル

モデルは大別すると三つの情報を持っています。

- **ステータス情報**
- **パーマネントアイテムデータベース**
- **テンポラリアイテムデータベース**

ステータス情報は、円の半径などのモーダルパラメータやスクリーンレイアウトなど、ユーザが変更した情報を保持します。新モデルではシステムの初期値が設定されます。これらの情報はモデルファイルに保存します。

パーマネントアイテムデータベースはアイテムを保持します。
新モデルではパーマネントアイテムデータベースは空でアイテムは 1 つもありません。パーマネントアイテムデータベースはモデルファイルに保存します。パーマネントアイテムデータベースは UNDO をサポートする機構を持ちます。

テンポラリアイテムデータベースはアイテムを作成しパーマネントアイテムデータベースに保存する時使用する一時的なデータベースです。これはモデルファイルに保存しません。

テンポラリアイテムデータベースのアイテムをテンポラリアイテム、パーマネントアイテムデータベースのアイテムをパーマネントアイテムと呼びます。両者の違いはパーマネントアイテムは参照だけに限り、直接変更を加えることができないことです。パーマネントアイテムを変更するには、一旦テンポラリアイテムにし、テンポラリアイテムを変更し、パーマネントデータベースを更新するという手順を踏みます。この方法はいくつか有利な点があります。アイテムの作成や変更を止めたいときは、テンポラリアイテムを破棄するだけで済みます。またパーマネントデータベースの更新を一元化することで、不注意によるパーマネントデータベースの破壊を防止できます。

3.1.1　アイテム識別子

パーマネントアイテムデータベースのアイテムには作成順に 1 から始まる番号が自動的に割り付けられます。この番号をアイテム識別子と呼びます。
アイテム識別子は一度割り付けると変わりません。アイテムのデータを変更してもアイテム識別子は変わりません。アイテムを削除したときは空き番になるだけで、この

アイテム識別子を他のアイテムに割り振ることはありません。UNDO コマンドを使って削除されたアイテムを復活させても以前のアイテム識別子を保持しています。アイテム識別子はパーマネントアイテムを特定するのに使います。
モデルをファイルに保存すると、削除されたアイテムを取り除き、番号を振り直します。このことから、アイテム識別子の有効期間はモデルをファイルに保存する迄です。

3.1.2 アイテム属性

アイテムは下記の属性を持っています。

- **アイテムの属するピクチャ番号**
- **アイテムタイプ**
- **クラス番号**
- **レビジョン番号**
- **線種**
- **線幅 など**

3.1.3 アイテムの構造

アイテムは、いくつかの「サブレコード」とよばれる単位データの並びで記述します。アイテムタイプによってどのサブレコードをどのような順序で記述するか決まっています。アイテムタイプごとのサブレコードの並びの規則、サブレコードの種類とその詳細は後の章で説明しますが、ここで単純なアイテムについて見てみましょう。
たとえば、線分アイテムではつぎのようになります。

順番	サブレコードタイプ	内容
#0	AwSr::STARTPOINT	**線分始点**
#1	AwSr::LINE	**線分終点**
#2	AwSr::EOI	**アイテムの終了**

最初に線分の始点をもつサブレコード、次に線分の終点を持つサブレコードがきます。最後にアイテムの終了を表すサブレコードをおきます。円／円弧アイテムの場合はつぎのようになります。

順番	サブレコードタイプ	内容
#0	AwSr::STARTPOINT	**円弧始点**
#1	AwSr::CIRCLE	**円弧データ**
#2	AwSr::EOI	**アイテムの終了**

最初に円弧始点をもつサブレコード、次に円弧データを持つサブレコードがきます。円弧始点は線分始点と同じサブレコードタイプ (AwSr::STARTPOINT) を使います。やはり最後にアイテムの終了を表すサブレコードをおきます。

3.1.4 モデルの実装

モデルとモデルを構成している要素は C++ クラスで実装しています。以下にクラス名を示します。

AwModel	**モデル**
AwPermItemDb	**パーマネントアイテムデータベース**
AwTempItemDb	**テンポラリアイテムデータベース**
AwItem	**アイテム（抽象クラス）**

AwTempItem	テンポラリアイテム（AwItem を継承）
AwItemAttributes	アイテム属性
AwSr	サブレコード（抽象クラス）
AwSrStart	始点サブレコード（AwSr を継承）
AwSrLine	線分サブレコード（AwSr を継承）
AwSrCircle	円・円弧サブレコード（AwSr を継承）
AwSrEoi	アイテム終端サブレコード（AwSr を継承）

アクティブモデルを得るには次のようにします。

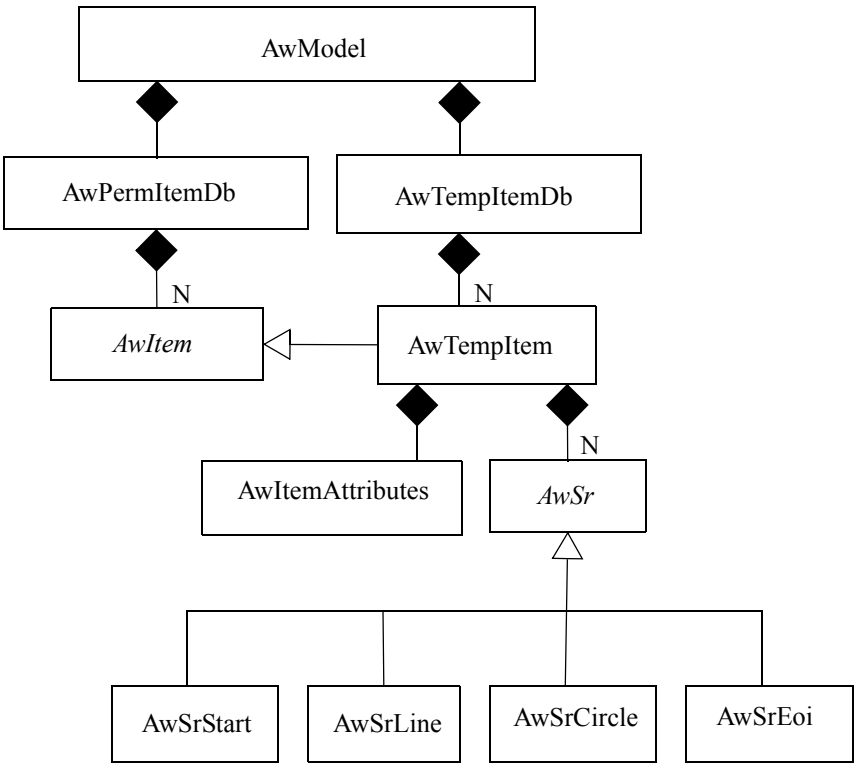
```
AwModel* pModel = GetActiveModel();
```

AwModel オブジェクトからパーマネントアイテムデータベースとテンポラリアイテムデータベースを得ます。

```
AwPermItemDb* pPermItemDb = pModel->GetPermItemDb();
AwTempItemDb* pTempItemDb = pModel->GetTempItemDb();
```

パーマネントアイテムデータベース AwPermItemDb オブジェクトからパーマネントアイテムを得ます。テンポラリアイテムデータベース AwTempItemDb オブジェクトからテンポラリアイテムを得ます。
テンポラリアイテム AwTempItem クラスは AwItem クラスを継承しています。

アイテムオブジェクトは AwItemAttributes オブジェクトと複数の AwSr オブジェクトを持ちます。アイテムからサブレコードオブジェクトを得ます。
AwSrStart、AwSrLine、AwSrCircle、AwSrEoi クラスは AwSr クラスを継承しています。他にも多くの種類のサブレコードクラスがあります。



最上位の AwModel オブジェクトから下位のオブジェクトを順番にたどって行くことができるのがわかると思います。このとき得られるオブジェクトのポインタについての注意があります。ポインタを使ってオブジェクトを直接 delete してはいけません。上位のオブジェクトが下位のオブジェクトの所有者で、

オブジェクトを管理しています。上位のオブジェクトには下位のオブジェクトを削除するメソッドがあります。もうひとつの注意点は、グローバル変数、スタティック変数などにオブジェクトへのポインタを保持してはならないということです。AwModel オブジェクトに変更があり、モデルを構成しているオブジェクトの一部が削除されることがあるからです。グローバル変数が保持しているポインタは無効になっているかもしれないのです。

3.1.5 データベースのサイズの上限

パーマネントアイテムデータベースに保持できるアイテム数やアイテムのサイズには制限があります。アプリケーションは起動時にコンフィグレーションファイルの記述を元にデータベースの最大値を設定します。アプリケーション実行中はこの最大値を変更することはできません。最大値を得るには下記のメソッドを使います。
アイテム数の最大値
```
AwPermItemDb::GetMaxItemCount();
```
アイテムのサブレコード数の最大値
```
AwItem::GetMaxSrCount();
```
アイテムのデータサイズの最大値（アイテムの持つサブレコードの合計サイズ。バイト）
```
AwItem::GetMaxDataSize();
```

現在設定可能な下限値で、全てのアイテムのデータサイズが最大値であると仮定した場合の、合計サイズは 16GB になります（設定可能な上限値で計算すればもっと大きな値になります)。これは 32bit アプリケーションが使えるプロセスのメモリ空間サイズ 2GB をはるかに超えます。実際には点、線分、円弧などサイズの小さなアイテムが多いので、このようなことにはなりませんが、可能性は排除できません。64bit アプリケーションではメモリ空間サイズが大きいので原理的にはこの問題は解消します。

モデルが持つ実際のアイテム数を知るには AwPermItemDb オブジェクトのメソッドを使います。
現在のアイテム識別子の最大値（削除アイテムも含むアイテム数と同じ）
```
GetMaxItemId();
```
現在のアイテム数（削除アイテムを除いた生きているアイテムの数）
```
GetItemCount();
```

これらの値の間には下記の関係が成立します。
```
0 <= pPermItemDb->GetItemCount() <= pPermItemDb->GetMaxItemId() <= AwPermItemDb::GetMaxItemCount()
```

3.2 パーマネントアイテムの参照

パーマネントアイテムへのポインタを得ると、パーマネントアイテムの持つ全てのデータを参照することができます。

3.2.1 アイテムを得る

パーマネントアイテムを得るには AwPermItemDb オブジェクトの GetItem() メソッドを使います。アイテム識別子かアイテム名を与えてパーマネントアイテムのポインタを得ます。アイテムが存在しなければ null ポインタを返します。
```
const AwItem* GetItem(int idptr);
const AwItem* GetItem(const std::string& name);
```

パーマネントアイテムへのポインタは const AwItem* ですから、パーマネントアイテムを変更することはできません。

3.2.2 アイテムへのアクセス

アイテムのポインタを得れば AwItem オブジェクトが持つアイテム属性やアイテムを構成しているサブレコードを参照することができます。次に示すコードフラグメントは線分アイテムのサブレコードを先頭から順番にアクセスします。

```
const AwItem* pItem = pModel->GetPermItemDb()->GetItem(idptr);
if (pItem == NULL)
  return;
if (pItem->GetType() != AwItem::LINE)
  return;
for (int i = 0; i < pItem->GetSrCount(); ++i) {
  const AwSr* pSr = pItem->GetSrAt(i);
  // 省略
}
```

GetSrAt() メソッドで得るサブレコードへのポインタは const AwSr* ですから、サブレコードを変更することはできません。

GetSrAt() メソッドで得た AwSr オブジェクトの生存期間は次の GetSrAt() メソッドの直前までという制限があります。次に示すコードフラグメントは誤りです。

```
const AwSr* pSr1 = pItem->GetSrAt(0);
const AwSr* pSr2 = pItem->GetSrAt(1);
```

変数 pSr2 のステートメントを実行した後では、変数 pSr1 の指すポインタは無効になります。これは現在の実装上の制限で将来取り除きたいところです。テンポラリアイテムオブジェクトにはこの制限はありません。

3.3 テンポラリアイテムデータベース

AwTempItemDb はテンポラリアイテム AwTempItem を管理するクラスです。テンポラリアイテムの作成、削除を行い、テンポラリアイテムをパーマネントアイテムデータベースに保存します。

テンポラリアイテム数の上限は 256 です。上限を超えてテンポラリアイテムを作成しようとすると、全てのテンポラリアイテムをパーマネントアイテムデータベースに保存し、クリアしてから、新しいテンポラリアイテムを作成します。

テンポラリアイテムの ID は 1 から始まる番号で指定します。

```
AwTempItemDb* pTempItemDb = pModel->GetTempItemDb();
for (int tid = 1; tid <= pTempItemDb->GetItemCount(); ++tid) {
  AwTempItem* pTempItem = pTempItemDb->GetItem(tid);
  for (int i = 0; i < pTempItem->GetSrCount(); ++i) {
    AwSr* pSr = pTempItem->GetSrAt(i);
    // 省略
  }
}
```

AwTempItemDb オブジェクトの GetItem() メソッドで得る AwTempItem オブジェクトへのポインタは non-const ポインタですから、テンポラリアイテムを変更できます。
AwTempItem オブジェクトの GetSrAt() メソッドで得るサブレコードへのポインタは non-const ポインタですから、サブレコードを変更できます。

AwTempItemDb オブジェクトが管理するテンポラリアイテムは、画面の再表示などで自動的にスクリーンに表示されます。画面の再表示を待たずに直ちに表示したいときは Tmpgrptn() 関数を使います。

```
Tmpgrptn(1, pTempItemDb->GetItemCount(), 0, 0);
```

第 1 引数、第 2 引数は表示する最初と最後のテンポラリアイテムの ID です。

3.4 アイテムの作成

新しいアイテムの作成は次の手順で行います。

- 空のテンポラリアイテムを作る。
- テンポラリアイテムにアイテムを構成するサブレコードを追加する。
- テンポラリアイテムをパーマネントアイテムデータベースに保存する。

線分アイテムを作るコードフラグメントを見てみましょう。線分の端点座標は (0,0)、(0,100) です。

```
AwTempItemDb* pTempItemDb = GetActiveModel()->GetTempItemDb();
pTempItemDb->Init();
AwTempItem* pTempItem = pTempItemDb->OpenItem();
if (pTempItem == NULL)
  return;
pTempItem->SetType(AwItem::LINE);
pTempItem->SetLineFont(1);
if (pTempItem->AddSr(AwSrStart(G2Point(0.0, 0.0))) &&
    pTempItem->AddSr(AwSrLine(G2Point(0.0, 100.0))) &&
    pTempItem->Close()) {
  pTempItemDb->StoreItems();
}
pTempItemDb->Init();
```

まず、テンポラリアイテムデータベースを得ます。テンポラリアイテムデータベースを空にします。

```
AwTempItemDb* pTempItemDb = GetActiveModel()->GetTempItemDb();
pTempItemDb->Init();
```

次に空のテンポラリアイテムを作成します。

```
AwTempItem* pTempItem = pTempItemDb->OpenItem();
if (pTempItem == NULL)
  return;
```

OpenItem() メソッドはテンポラリアイテム作成に失敗したときは null ポインタを返します。成功したときは AwTempItem オブジェクトへのポインタを返します。このアイテムのアイテム属性は「現在の値」が設定されますので、それでよければ変更しません。アイテムタイプだけは必ず設定しなければなりません。アイテムタイプを表す定数は AwItem クラスの enum で定義していますのでそれを使います。

```
pTempItem->SetType(AwItem::LINE);
pTempItem->SetLineFont(1);
```

空のアイテムに線分アイテムを構成するサブレコードを追加します。ここでは線分の始点 AwSrStart と線分の終点 AwSrLine の 2 つのサブレコードを追加します。AddSr() メソッドは成功したら true を返します。

```
pTempItem->AddSr(AwSrStart(G2Point(0.0, 0.0)));
pTempItem->AddSr(AwSrLine(G2Point(0.0, 100.0)));
```

最後に Close() メソッドでアイテムをクローズします。このメソッドはアイテムに AwSrEoi サブレコードを追加します。メソッドは成功したら true を返します。

```
pTempItem->Close();
```

以上が成功したらテンポラリアイテムをパーマネントアイテムデータベースに保存します。成功するとテンポラリアイテムは削除されます。

```
  pTempItemDb->StoreItems();
```

サブレコードの追加に失敗したときは、このテンポラリアイテムを削除します。テンポラリアイテムデータベースを空にするのが最も簡単な方法です。

```
pTempItemDb->Init();
```

次のようにしても同じ効果があります。

```
pTempItemDb->RemoveLastItem();
```

3.5 アイテムの変更

パーマネントアイテムの変更は次の手順で行います。

- パーマネントアイテムをコピイしてテンポラリアイテムを作る。
- テンポラリアイテムを変更する。
- テンポラリアイテムをパーマネントアイテムデータベースに保存する。

テンポラリアイテムはサブレコードの追加、挿入、削除ができます。また、サブレコードの内容を直接変更することができます。

線分アイテムを Y 軸方向へ 20 移動するコードフラグメントを見てみましょう。これはサブレコードの内容を直接変更する例です。

```
// item は線分アイテムへの参照 (const AwItem&) とします。
AwTempItemDb* pTempItemDb = GetActiveModel()->GetTempItemDb();
pTempItemDb->Init();
AwTempItem* pTempItem = pTempItemDb->LoadItem(1, item);
if (pTempItem == NULL)
  return;
const G2Point vec(0.0, 20.0);
for (int i = 0; i < pTempItem->GetSrCount(); ++i) {
  AwSr* pSr = pTempItem->GetSrAt(i);
  pSr->Translate(vec);
}
pTempItemDb->StoreItems();
pTempItemDb->Init();
```

パーマネントアイテムをコピイしてテンポラリアイテムを作るには AwTempItemDb オブジェクトの LoadItem() メソッドを使います。

```
AwTempItem* pTempItem = pTempItemDb->LoadItem(1, item);
if (pTempItem == NULL)
  return;
```

第 1 引数は「パーマネントアイテムの更新」を表す 1 です。第 2 引数はパーマネントアイテムへの参照を渡します。ここでは変数 item は線分アイテムへの参照だと思って下さい。LoadItem() メソッドはテンポラリアイテム作成に失敗したときは null ポインタを返します。成功したときは AwTempItem オブジェクトへのポインタを返します。
次の for ループはアイテムのサブレコードを先頭から順番にアクセスしています。これはパーマネントアイテムの場合と同じです。AwTempItem オブジェクトの GetSrAt() メソッドの返す AwSr ポインタは non-const で、サブレコードの内容を直接変更できるところが大きな違いです。

```
const G2Point vec(0.0, 20.0);
for (int i = 0; i < pTempItem->GetSrCount(); ++i) {
  AwSr* pSr = pTempItem->GetSrAt(i);
  pSr->Translate(vec);
}
```

AwSr の Translate() メソッドは AwSrStart、AwSrLine オブジェクトの座標を引数 vec で指示した分移動します。AwSrEoi オブジェクトは座標値を持たないので AwSrEoi の Translate() メソッドは何もしません。同じ Translate() メソッドですが、クラスのごとに適切に機能を果たすようになっています。このようなメソッドはサブレコードタイプを考慮する必要がないという利点があります。

最後にテンポラリアイテムをパーマネントアイテムデータベースに保存します。LoadItem() メソッドで作成したテンポラリアイテムは最後に AwSr::EOI サブレコードを持っていますので Close() メソッドは使いません。

```
pTempItemDb->StoreItems();
```

次のコードフラグメントはサブレコードを挿入する例です。線分アイテムの端点は (0,20),(100,20) で、3 つのサブレコードを持っています (AwSrStart、AwSrLine、AwSrEoi)。この線分の (40,20) から (60,20) の間を破線にします。(40,20) と (60,20) の 2 つの線分サブレコードを挿入します。

```
// item は線分アイテムへの参照 (const AwItem&) とします。
AwTempItemDb* pTempItemDb = GetActiveModel()->GetTempItemDb();
pTempItemDb->Init();
AwTempItem* pTempItem = pTempItemDb->LoadItem(1, item);
if (pTempItem == NULL)
  return;
if (pTempItem->InsertSrAt(1, AwSrLine(G2Point(40.0, 20.0))) &&
    pTempItem->InsertSrAt(2, AwSrLine(G2Point(60.0, 20.0), 3))) {
  pTempItemDb->StoreItems();
}
pTempItemDb->Init();
```

InsertSrAt() メソッドはテンポラリアイテムオブジェクトにサブレコードを挿入します。第 1 引数がサブレコードの挿入位置を示すインデックスです。インデックスで示された位置とそれに以降にあるサブレコードを後方に移動し、インデックスの位置にサブレコードを追加します。2 つめの挿入サブレコードはサブレコードの線種を 3（破線を表す番号）にします。これで、1 本の線分は 3 つの部分に分割されました。3 つの部分は、折れ曲がったり Z 字になったりしてはならず、1 本の線分を構成しなければなりません。

3.6 ドラフティングスケール

グラフィックステキストの文字高さ、マークサイズなどはプリントアウトした紙面上の大きさを指定します。図面には縮尺があり、そこには部分拡大図も記入します。こうした時、図面上の文字の大きさを統一するには、図面上での文字の大きさを指定するのが良いという考え方です。そうでないと、図面縮尺と部分拡大図倍率を考慮した文字の大きさを指定しなければなりません。たとえば図面上の文字の大きさ 3mm は、図面縮尺 1/5 なら 15mm、図面縮尺 1/200 なら 600mm としなければなりません。図面上での文字の大きさを指定するのであれば簡単です。

ドローイングスケール が図面縮尺、ピクチャスケールが部分拡大図倍率に相当します。
ここでドラフティングスケールを次式で定義します。

ドラフティングスケール＝１／（ピクチャスケール＊ドローイングスケール）

これは、図面上の文字の大きさをモデルスペースでの大きさに変換する倍率です。
ピクチャスケール PSF=2.0、ドローイングスケール DSF=1/5 とすれば、ドラフティングスケールは 2.5 となります。

SCALE = 1.0 / (PSF*DSF) = 1.0 / (2.0 * 0.2) = 2.5

文字高さ h=4.0 とすれば、モデルスペースでの文字高さ H は 10.0 になります。

H = h * SCALE = 4.0 * 2.5 = 10.0

ドラフティングスケールは AwModel オブジェクトの GetDraftingScale() メソッドで得ることができます。引数 pid にはピクチャ番号を与えます。

```
double scale = pModel->GetDraftingScale(pid);
```

もっと使いやすいのは AwItem オブジェクトの GetDraftingScale() メソッドでしょう。AwItem オブジェクトのピクチャ番号を使ってドラフティングスケールを計算しますので、引数なしです。

```
double scale = pItem->GetDraftingScale();
```

次のコードフラグメントはテンポラリアイテムに文字列サブレコードを追加する部分です。文字列サブレコードの SetLocation() メソッドの第 2 引数にドラフティングスケールを渡します。

```
const AwDrafParam* pDrafParam = pModel->GetDrafParam();
AwSrText srText;
srText.SetTextParams(*pDrafParam);
srText.SetText("Hello world.");
```

```
srText.SetLocation(G2Point::Origin, pTempItem->GetDraftingScale());
if (! pTempItem->AddSr(srText))
  return false;
```

AwSrText オブジェクトの SetLocation() メソッドは文字列の外形を計算します。外形はモデルスペース座標なのでドラフティングスケールを使って計算します。この外形は表示するものではなく、主にアイテムのピックなどが使用します。ドローイングスケール やピクチャスケールを変更しても、この外形はそのままで追従しないので、不都合があります。

3.7 データベースイテレータ

パーマネントアイテムデータベースから、特定のピクチャ番号や特定のアイテムタイプのアイテムだけを順次取り出したい場合があります。このようなときは AwPermDbIterator クラスを使います。ピクチャ 1 のアイテムを順番に取り出すには次のようにします。

```
std::auto_ptr<AwPermDbIterator> apIterator(pModel->CreateDbIterator());
if (apIterator.get() == NULL)
  return;
apIterator->SelectPicture(1);
while (apIterator->HasNext()) {
  const AwItem* pItem = apIterator->NextItem();
  // 省略
}
```

AwPermDbIterator オブジェクトを得るには AwModel オブジェクトの CreateDbIterator() メソッドを使います。このメソッドは AwPermDbIterator オブジェクトを生成しそのポインタを返します。このオブジェクトは不要になった時点で delete しなければなりませんが、この例では std::auto_ptr を使用しているので、apIterator オブジェクトのデストラクタで自動的に delete されます。
AwPermDbIterator オブジェクトの SelectPicture() メソッドで対象ピクチャ番号を指定します。ここではピクチャ 1 を指定しています。AwPermDbIterator オブジェクトは複数のピクチャを指定するメソッドも持っています。

アイテムの取り出しを行っている部分は次のようになっています。

```
while (apIterator->HasNext()) {
  const AwItem* pItem = apIterator->NextItem();
  // 省略
}
```

while ループの条件式の HasNext() メソッドが次のアイテムがあるかを調べます。メソッドは次のアイテムがあるとき true を返します。true のときは NextItem() メソッドで次のパーマネントアイテムのポインタを得ます。次に HasNext() メソッドが呼ばれるまで、NextItem() メソッドは同じものを返します。

線分アイテムだけを取り出すなら、次のコードフラグメントを while ループの前に挿入します。

```
AwSelectableMask* pMask = apIterator->GetMask();
pMask->SetSelectableAll(AwItemAttributes::BY_ITEMTYPE, false);
pMask->SetSelectable(AwItemAttributes::BY_ITEMTYPE, AwItem::LINE, true);
apIterator->SetMaskEnabled(true);
```

AwPermDbIterator オブジェクトが持つ AwSelectableMask オブジェクトの選択するアイテムタイプを線分だけにします。

もうひとつは AwItemIdsIterator クラスです。このイテレータはアクティブリストやハイライトアイテムリストなどのアイテム識別子の配列をイテレートします。先ほど説明した AwPermDbIterator クラスと AwItemIdsIterator クラスは AwItemListIterator クラスを継承していて、HasNext()、NextItem() メソッドを持ち、それらのメソッドの使い方はまったく同じです。

下記に AwItemIdsIterator クラスのコンストラクタを示します。

```
AwItemIdsIterator(const AwPermItemDb* pItemDb, const AwItemIdArray& itemIds);
```

```
AwItemIdsIterator(const AwPermItemDb* pItemDb, int count, const int itemIds[]);
AwItemIdsIterator(const AwPermItemDb* pItemDb, int first, int last);
```

最初のコンストラクタはアイテム識別子を持つ AwItemIdArray オブジェクトを受け取り、2 番目はアイテム識別子を持つ int 配列を受け取ります。3 番めのコンストラクタは、イテレートする最初と最後のアイテム識別子を受け取ります。コンストラクタに渡した AwItemIdArray オブジェクトや int 配列の内容をイテレート途中で変更してはいけません。

AwPermDbIterator クラスの例のように、ピクチャ 1 の線分アイテムを順番に取り出すには次のように記述します。HasNext() メソッドはアイテム識別子で特定するアイテムが存在しないときは、それを無視して次へ進みます。

```
AwItemIdsIterator iterator(pPermItemDb, 1, pPermItemDb->GetMaxItemId());
while (iterator.HasNext()) {
  const AwItem* pItem = iterator.NextItem();
  if (pItem->GetPictureId() != 1 || pItem->GetType() != AwItem::LINE)
    continue;
  // 省略
}
```

この場合は while ループ内でアイテムのピクチャ番号とアイテムタイプを判定しなければならないことに注意してください。AwPermDbIterator クラスの場合は事前に条件を設定するだけで済む利点があります。

第 4 章　モデル

アプリケーションのデータを管理するクラスを説明します。このクラスでは多くのオブジェクトを操作しますが以下の点に留意してください。

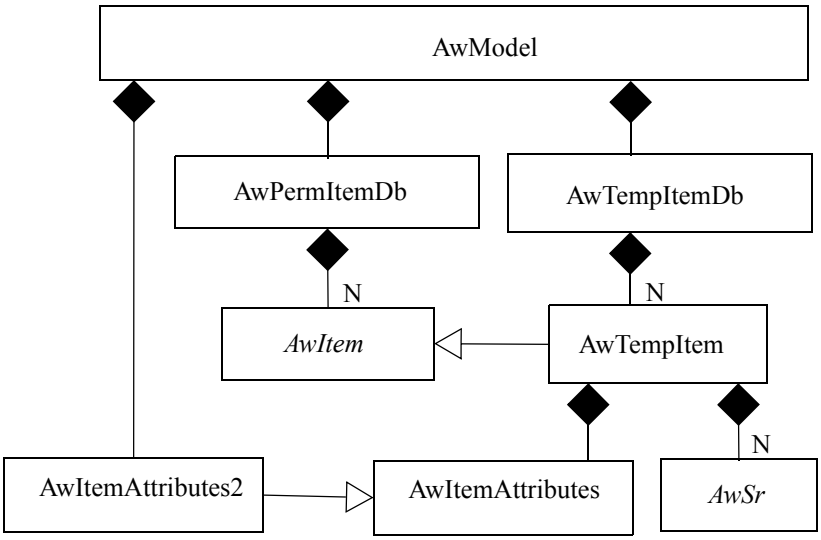
多くのクラスは所有するオブジェクトを取得するメソッドを提供します（Getter と呼びます)。Getter で取得したオブジェクトのポインタを使ってオブジェクトを直接 delete してはなりません。
Getter で取得したオブジェクトのポインタをグローバル変数やスタティック変数に保持してはなりません。オブジェクトは自身が管理しているオブジェクトを削除することがあります。その結果グローバル変数などが保持しているポインタが無効となります。

クラスを継承してサブクラスを作ってはなりせん。

4.1　AwModel クラス

このクラスはアプリケーションのデータを管理する最上位のクラスです。このクラスは自身が所有するオブジェクトを取得するメソッドを提供します。モデルオブジェクトとその配下のオブジェクトは、データの追加、更新、削除などの操作を行いますが、この操作の結果をスクリーンに反映させるなどの表示は行いません。

下記はこれから説明する主なクラスを含むクラス図です。



4.1.1 オブジェクトを得る

オブジェクト取得メソッドには const と non-const の 2 種類のメソッドがあるものがあります。
const メソッドは オブジェクトの const ポインタを返しますので、取得したオブジェクトの const メソッドだけが使えます。オブジェクトは読み出し専用で変更はできません。const メソッドは取得したいオブジェクトがないと null ポインタを返します。
一方 non-const メソッドは、取得したいオブジェクトをがないとそれを生成し (new)、その non-const ポインタを返します。取得したオブジェクトの non-const メソッドを使ってオブジェクトを変更することができます。

パーマネントアイテムデータベースを得ます。
```
const AwPermItemDb* GetPermItemDb() const;
AwPermItemDb* GetPermItemDb();
```
AwPermItemDb オブジェクトへのポインタを返します。

テンポラリアイテムデータベースを得ます。
```
AwTempItemDb* GetTempItemDb();
```
AwTempItemDb オブジェクトへのポインタを返します。

アイテム属性の現在値を得ます。
```
const AwItemAttributes2* GetItemAttributes() const;
AwItemAttributes2* GetItemAttributes();
```
AwItemAttributes2 オブジェクトへのポインタを返します。

幾何演算定数を得ます。
```
const AwGeomParam* GetGeomParam() const;
AwGeomParam* GetGeomParam();
```
AwGeomParam オブジェクトへのポインタを返します。

製図用定数を得ます。
```
const AwDrafParam* GetDrafParam() const;
AwDrafParam* GetDrafParam();
```
AwDrafParam オブジェクトへのポインタを返します。

一時的なアイテム選択マスクを得ます。
```
AwSelectableMask* GetTemporarySelectableMask();
```
AwSelectableMask オブジェクトへのポインタを返します。
アイテム選択マスクを得ます。
```
AwSelectableMask* GetSelectableMask();
```
AwSelectableMask オブジェクトへのポインタを返します。

モデルタイトルセットを得ます。
```
AwModelTitleSet* GetModelTitleSet();
```
AwModelTitleSet オブジェクトへのポインタを返します。

アイテム属性テーブルを得ます。
```
const AwItemAttrTable* GetItemAttrTable() const;
AwItemAttrTable* GetItemAttrTable();
```
AwItemAttrTable オブジェクトへのポインタを返します。

ピクチャリストを得ます。
```
const AwPictureList* GetPictureList() const;
AwPictureList* GetPictureList();
```
AwPictureList オブジェクトへのポインタを返します。

ドローイングレイアウトリストを得ます。
```
const AwDloList* GetDloList() const;
```

```
AwDloList* GetDloList();
```
AwDloList オブジェクトへのポインタを返します。

プロッタのペン割付けを得ます。
```
AwPenAssignment* GetPenAssignment();
```
AwPenAssignment オブジェクトへのポインタを返します。

「その他のパラメータ」を得ます。
```
const AwMiscParam* GetMiscParam() const;
AwMiscParam* GetMiscParam();
```
AwMiscParam オブジェクトへのポインタを返します。

4.1.2 ドラフティングスケール

ドラフティングスケールは次式で定義します。
ドラフティングスケール＝１／（ピクチャスケール＊ドローイングスケール）
これは、グラフィックステキストの文字高さ、マークサイズなどの図面上（あるいは紙面上）の大きさをピクチャ上での大きさに変換する倍率です。ピクチャ上での実際の文字高さは次式で計算できます。
ピクチャでの文字高さ ＝ 図面上の文字高さ＊ドラフティングスケール

指定したピクチャのドラフティングスケールを得ます。
```
double GetDraftingScale(int pid) const;
```
pid　　ピクチャ番号。
戻り値　ドラフティングスケール。

4.1.3 データベースイテレータ

パーマネントアイテムを順次取り出すにはイテレータを使うのが便利です。イテレータを表現するクラス AwPermDbIterator の詳細は後で説明します。

次のメソッドは新しいイテレータオブジェクトを作成します。
```
AwPermDbIterator* CreateDbIterator() const;
```
AwPermDbIterator オブジェクトを作成し、そのオブジェクトのポインタを返します。このメソッドで得たオブジェクトは不要になった時点で削除（delete）しなければなりません。

4.2 AwPermItemDb クラス

このクラスはパーマネントアイテムデータベースを表現します。パーマネントアイテムデータベースはパーマネントアイテムを保持し、アイテムの追加、削除という重要な役割を果たします。
AwPermItemDb オブジェクトは AwModel オブジェクトから得ます。

4.2.1 アイテム数の上限

パーマネントアイテムデータベースに格納できるアイテム数には上限があります。この値はアプリケーション起動時に一度だけ設定されます。その後は変更してはなりません。この値は全ての AwPermItemDb オブジェクトに共通です。

```
static int GetMaxItemCount();
```
アイテム数の上限値を返します。

4.2.2 アイテム数

このオブジェクトが持つアイテム数を得る 2 つのメソッドがあります。

```
int GetMaxItemId() const;
```
アイテム識別子の最大値を返します。
```
int GetItemCount() const;
```
アイテム数を返します。

GetMaxItemId() メソッドはこのオブジェクトに格納されているアイテム識別子の最大値を返します。アイテムがひとつもなければ 0 です。このメソッドが返す値は最後に作成されたアイテムの識別子です。これはそのアイテムが削除されても変わりません。次に作成するアイテムのアイテム識別子はこのメソッドが返すアイテム識別子に 1 を加えたものになります。アイテム識別子の値は常に 1 づつ増加します。アイテム識別子はアイテム数の上限を超えることはできません。

GetItemCount() メソッドは削除アイテムを除いた生きているアイテムの数を返します。アイテムがひとつも削除されていなければ、アイテム識別子の最大値と同じです。

これらの値の間には下記の関係が成立します。
```
0 <= GetItemCount() <= GetMaxItemId() <= AwPermItemDb::GetMaxItemCount()
```

4.2.3 アイテムの取得

このオブジェクトにはパーマネントアイテムを得る 2 つのメソッドがあります。アイテム識別子またはアイテム名を指定してパーマネントアイテムオブジェクトへのポインタを得ます。これらのメソッドは AwItem オブジェクトへの const ポインタを返します。const ポインタなので、取得したオブジェクトの const メソッドだけが使えます。パーマネントアイテムオブジェクトは読み出し専用で、変更はできません。

指定したアイテム識別子のアイテムを得ます。
```
const AwItem* GetItem(int idptr) const;
```
idptr　アイテム識別子（1 <= idptr）。
戻り値　アイテム識別子で指定したパーマネントアイテムオブジェクトへのポインタを返します。アイテム識別子で指定したアイテムが存在しないか削除されているなら null ポインタを返します。

指定したアイテム名のアイテムを得ます。
```
const AwItem* GetItem(const std::string& name) const;
```
name　アイテム名。
戻り値　指定したアイテム名を持つパーマネントアイテムオブジェクトへのポインタを返します。指定したアイテム名を持つアイテムが存在しないか削除されているなら null ポインタを返します。

4.2.4 アイテムの削除

このオブジェクトにはパーマネントアイテムを削除する 2 つのメソッドがあります。これらのメソッドはオブジェクトからアイテムを削除しますが、スクリーンの更新は行いません。アイテム削除と共にスクリーンの更新を行うには Dbdlitems()、Dbdlitmpc() 関数を使います。これらの関数の詳細は後ほど説明します。

指定したアイテムを削除します。
```
bool DeleteItem(int key_ud, int idptr);
```
key_ud　1 を渡します。
idptr　アイテム識別子（1 <= idptr）。
戻り値　アイテム識別子で指定したパーマネントアイテムを削除したら true を返します。

アイテム識別子の配列を使って複数のアイテムを削除します。
```
int  DeleteItems(int key_ud, const AwItemIdArray& itemIds);
```

key_ud　1 を渡します。
itemIds　アイテム識別子を持つ AwItemIdArray オブジェクトの参照。負のアイテム識別子があるとその絶対値を使います。
戻り値　実際に削除したアイテム数を返します。

4.2.5　UNDO ブロック

UNDO は指定したパーマネントアイテムをひとつ前の状態に戻す機能です。例えば、位置を移動したアイテムを移動前の位置に戻すとか、削除されたアイテムを復活させるなどです。UNDO はアイテム単位で処理するか、複数のアイテムを含むブロックと呼ぶ単位で処理します。あるコマンドで一度に複数のアイテムを処理した場合は UNDO ブロック単位で戻すのが便利なことがあります。

このオブジェクトは UNDO ブロックを制御するメソッドを持っています。
```
void NewUndoBlock();
```
UNDO ブロックを区切ります。

通常、コマンドハンドラがコマンドエンドトークン <CE> を処理した後、このメソッドを呼出し現在の UNDO ブロックを終了させます。これは直ちに新しい UNDO ブロックの開始になります。この標準の UNDO ブロック制御よりも詳細な制御をしたいときに NewUndoBlock() メソッドを使います。

UNDO 操作を行うには DbUundo() 関数を使います。この関数は、UNDO 操作と共にスクリーンの更新も行います。関数の詳細は後ほど説明します。

4.2.6　アイテム名

パーマネントアイテムにはユニークな名前を付けることができます。名前は英字で始まり英字か数字が続く次の形式を薦めます。空白を含めることはできません。

[A-Z][A-Z0-9]*

名前の長さは AwItemAttributes::MAX_NAME_LENGTH 以下でなければなりません。名前は重複できません。アイテム名を持つアイテムを削除しても名前は消えず、その名前を他のアイテムに付けることはできません。UNDO で削除されたアイテムが復活したときも以前の名前が使えるようにするためです。それを避けたければ、アイテムを削除する前にアイテム名を取り除いておくことです。アイテム名は UNDO の対象外で、UNDO しても以前のアイテム名には戻りません。

指定したアイテムに名前を付けます。既にアイテムに名前がついていれば新しい名前に変えます。
```
bool SetItemName(int idptr, const std::string& name);
```
idptr　アイテム識別子（1 <= idptr）。
name　アイテム名（空文字列は不可。名前の削除にはなりません）。
戻り値　アイテム名をつけたとき true を返します。

指定したアイテムの名前を削除します。
```
bool RemoveItemName(int idptr);
```
idptr　アイテム識別子（1 <= idptr）。
戻り値　アイテム名を削除したとき true を返します。

4.2.7　ピクチャの情報

このオブジェクトはピクチャに関連する情報を得るメソッドを持ちます。これ以外のピクチャの情報は AwPicture クラスから得ます。AwPicture クラスについては第 9 章で説明します。

指定したピクチャのアイテム数を得ます。
```
int GetItemCountOfPicture(int pid) const;
```
pid　ピクチャ番号。

戻り値　削除アイテムを除いた生きているアイテムの数を返します。

指定したピクチャに属す全てのアイテムを内部に包み込む最小の矩形を得ます。矩形は G2Rect クラスを使います。G2Rect クラスの詳細は第 11 章を参照してください。

```
G2Rect GetPictureBounds(int pid) const;
```

pid　　ピクチャ番号。

戻り値　ピクチャの最小矩形を返します。生きているアイテムがないときや座標を持つアイテムがないときは無効な矩形オブジェクトを返します。

ピクチャにアイテムがないことは正常な状態です。アイテムがないピクチャの最小矩形を得た場合、その矩形オブジェクトは有効なデータを持たないというだけです。

4.3　AwItem クラス

このクラスはアイテムを表現する抽象クラスです。パーマネントアイテムとテンポラリアイテムはこのクラスを継承しています。このクラスはパーマネントアイテムとテンポラリアイテムに共通なメソッドを持ちます。このクラスは const メソッドだけを持ち、オブジェクトは読み出し専用で変更することはできません。

4.3.1　アイテムサイズの上限

ひとつのアイテムが持てるサブレコード数とサブレコードの合計サイズの上限があります。これらはアプリケーション起動時に一度だけ設定されます。その後は変更してはなりません。この値は全ての AwItem オブジェクトに共通です。

```
static int  GetMaxSrCount();
```

アイテムのサブレコード数の上限を返します。

```
static int  GetMaxDataSize()
```

アイテムのサブレコードデータの合計サイズの上限を返します（単位はバイト）。

4.3.2　アイテム属性

アイテムはアイテム属性を持ちます。アイテム属性は後述する AwItemAttributes クラスを使います。
次のメソッドでアイテム属性を得ます。

```
const AwItemAttributes& GetAttributes() const;
```

AwItemAttributes オブジェクトの const 参照を返します。

AwItemAttributes オブジェクトから全てのアイテム属性を得ることができます。
以下は簡便のために用意したメッソドです。

```
int  GetType() const;
```

アイテムタイプを表す整数を返します。

```
int  GetClass() const;
```

アイテムのクラス番号を返します。

```
int  GetRevision() const;
```

アイテムのレビジョン番号を返します。

```
int  GetLineFont() const;
```

アイテムの線種番号を返します。

```
int  GetLineWeight() const;
```

アイテムの線幅番号を返します。

```
int  GetColor() const;
```

アイテムの表示色番号を返します。

```
bool IsBlanked() const;
```

アイテムのブランキングフラグを返します。アイテムがブランキング（表示しない）なら true を返します。

```
int  GetPictureId() const;
```

アイテムのピクチャ番号を返します。

このアイテムが曲線アイテムか判定します。曲線アイテムは AwItem::LINE、CIRCLE、SPLINE、STRING の総称です。
```
bool IsCurve() const;
```
このアイテムが曲線アイテムなら true を返します。

このアイテムが製図アイテムか判定します。製図アイテムは AwItem::TEXT、MARK、DIMESNION、GTOL、XHT、AFL の総称です。
```
bool IsDrafting() const;
```
このアイテムが製図アイテムなら true を返します。

4.3.3 Getter

次のふたつのメソッドは主にパーマネントアイテムに適用します。テンポラリアイテムの場合は後述する AwTempItem クラスを参照してください。
```
int GetItemId() const;
```
このアイテムのアイテム識別子を返します。
```
const std::string GetName() const;
```
このアイテムのアイテム名を返します。アイテム名がついていなければ空文字列を返します。

アイテムを内部に包み込む最小の矩形を得ます。矩形は G2Rect クラスを使います。G2Rect クラスの詳細は第 11 章を参照してください。
```
G2Rect GetBounds() const;
```
このアイテムの最小矩形を返します。座標を持たないアイテムでは矩形は計算できませんので無効な矩形オブジェクトを返します。

アソシエイトアイテムのような非表示のアイテムは座標を持たないので、アイテムの最小矩形オブジェクトは有効なデータを持ちません。次のコードフラグメントはアイテムの最小矩形が有効か判定する例です。
```
G2Rect rect = pItem->GetBounds();
if (! rect.IsValid())
  return false;
```

このアイテムが属するピクチャのドラフティングスケールを得ます。AwModel クラスの GetDraftingScale() メソッドを使っています。
```
double GetDraftingScale() const;
```
ドラフティングスケールを返します。

4.3.4 サブレコード取得

アイテムは複数のサブレコードを持ちます。それは抽象クラス AwSr オブジェクトの配列のようなものと考えることができます。

次のメソッドでアイテムの持つサブレコード数を得ます。
```
int GetSrCount() const;
```
このアイテムが持つサブレコード数を返します。

アイテムのサブレコードを取得するには次のメソッドを使います。取得するサブレコードを指定するには配列のようにインデックス（0 から始まる）を使います。サブレコードは任意の順序でアクセスできます。
```
const AwSr* GetSrAt(int index) const;
```
index　サブレコードの位置を示すインデックス。0 <= index < GetSrCount()。

戻り値　index の位置の AwSr オブジェクトへのポインタを返します。index が範囲外のときは null ポインタを返します。

次に示すコードフラグメントはアイテムの先頭から順次サブレコードにアクセスする例です。

```
const AwItem* pItem = pModel->GetPermItemDb()->GetItem(idptr);
if (pItem == NULL)
  return;
for (int i = 0; i < pItem->GetSrCount(); ++i) {
  const AwSr* pSr = pItem->GetSrAt(i);
  if (pSr->GetId() == AwSr::EOI) {
    break;
  } else if (pSr->GetId() == AwSr::CATEGORY) {
    const AwSrCategory* pSrCateg = static_cast<const AwSrCategory*>(pSr);
    // AwSrCategory クラスのメソッドを使うことができる。
  } else if (pSr->GetId() == AwSr::POINT) {
    const AwSrPoint* pSrPoint = static_cast<const AwSrPoint*>(pSr);
    // AwSrPoint クラスのメソッドを使うことができる。
  }
}
```

上記の例では、for() ループ内の次の行でサブレコードオブジェクトを得ています。

```
const AwSr* pSr = pItem->GetSrAt(i);
```

pSr はサブレコードを表す抽象クラス AwSr へのポインタです。このオブジェクトのサブレコードタイプを知るには GetId() メソッドを使います。オブジェクトのサブレコードタイプがわかると、pSr ポインタを具象サブレコードクラスのポインタにダウンキャストすることができます。
具象サブレコードクラスについては第 6 章を参照してください。

4.3.5 サブレコード探索

アイテムが特定のタイプのサブレコードを持つかどうか調べたいことがあります。先に説明したようにアイテムのサブレコードを先頭から順番に調べればわかりますが、簡便のため次のようなメソッドがあります。

指定したタイプのサブレコードを探索します。

```
int SearchSr(int type, int first = 0, int last = AwItem::MAX_INDEX) const;
```

type	探索するサブレコードのタイプ。
first	サブレコードの探索開始位置。省略すると先頭から。
last	サブレコードの探索終了位置。省略すると最後まで。0 <= first <= last。
戻り値	最初に見つけたサブレコードのインデックスを返します。見つからなければ -1 を返します。

後方から前方に向かって指定したタイプのサブレコードを探索します。

```
int SearchSrBackward(int type, int last = AwItem::MAX_INDEX, int first = 0) const;
```

type	探索するサブレコードのタイプ。
last	サブレコードの探索開始位置。省略すると最後から。
first	サブレコードの探索終了位置。省略すると先頭まで。0 <= first <= last。
戻り値	最初に見つけたサブレコードのインデックスを返します。見つからなければ -1 を返します。

アイテムの曲線部分を調べたいことがあります。曲線部分とはサブレコードの並びが次のようになっている部分を言います。

```
AwSr::STARTPOINT, [{AwSr::LINE | AwSr::CIRCLE | AwSr::BEZIER}]+
```

曲線部分はサブレコードタイプ STARTPOINT で始まり、サブレコードタイプ LINE、CIRCLE、BEZIER のどれかがひとつ以上続くものです。この曲線部分には他のタイプのサブレコードが入ってはいけません。線分、円、スプラインなどの曲線アイテムは曲線部分をひとつだけ持っています。複合アイテムのように曲線部分を幾つも持つアイテムもあります。

曲線部分を探索します。

```
bool SearchCurve(int first, int last, int indices[]) const;
```

first　サブレコードの探索開始位置。省略すると先頭から。
last　サブレコードの探索終了位置。省略すると最後まで。0 <= first <= last。
indices　曲線部分を表すサブレコードの位置を格納する配列。曲線部分が見つかれば下記の値を設定します。
indices[0] : 曲線開始サブレコードの位置（AwSr::STARTPOINT）。
indices[1] : 曲線終了サブレコードの位置（AwSr::LINE、CIRCLE または BEZIER）。
first <= indices[0] < indices[1] <= last。
次の二つはスプラインアイテムのときだけ設定します。
トリムされたスプラインは片側あるいは両端に非表示のサブレコードを持っています。トリム部分を除いた曲線部分を表します。トリムしてなければ、indices[0]、indices[1] と同じ値です。
indices[2] : 最初の表示サブレコードの位置（AwSr::STARTPOINT または BEZIER）。
indices[3] : 最後の表示サブレコードの位置（AwSr::BEZIER）。
indices[0] <= indices[2] <= indices[3] <= indices[1]。
戻り値　曲線部分が見つかれば true を返します。

指定したサブレコードを含む曲線部分を探索します。
```
bool SearchCurve(int index, int indices[]) const;
```
index　サブレコードの位置。0 <= index < GetSrCount()。
indices　曲線部分を表すサブレコードの位置を格納する配列。曲線部分が見つかれば下記の値を設定します。0 <= indices[0] <= index <= indices[1] < GetSrCount()。
戻り値　曲線部分が見つかれば true を返します。

指定した位置のサブレコードから幾何図形を得ます。
得られる幾何図形はサブレコードの種類に依存します。

AwSr::POINT	G2Point オブジェクト
AwSr::LINE	G2Line オブジェクト
AwSr::CIRCLE	G2Circle オブジェクト
AwSr::BEZIER	G2Bezier オブジェクト

これ以外のサブレコードからは幾何図形を得ることはできません。幾何図形クラスの詳細は第 11 章を参照してください。

```
G2Geom* GetGeometryAt(int index) const;
```
index　サブレコードの位置を示すインデックス。0 <= index < GetSrCount()。
戻り値　index の位置のサブレコードから幾何図形オブジェクトを生成し、そのオブジェクトへのポインタを返します。index が範囲外のときやサブレコードから幾何図形オブジェクトを生成できなければ null ポインタを返します。このメソッドで得たオブジェクトは不要になった時点で削除（delete）しなければなりません。

4.4 AwItemAttributes クラス

このクラスはアイテム属性を表現するクラスです。アイテム属性には以下のものが含まれます。
- **アイテムタイプ**
- **クラス番号**
- **レビジョン番号**
- **線種**
- **線幅**
- **色番号**
- **ブランキングフラッグ**
- **アイテムの属するピクチャ番号**

このクラスのヘッダーファイルにはアイテム属性それぞれの上限値を定義した enum があります。
以下はアイテム属性を検査するクラスメソッドです。引数の値が適切であれば true を返します。上限値を使って判定するより簡便です。
```
static bool IsValidType(int type);
```

```
static bool IsValidClass(int cls);
static bool IsValidRevision(int rev);
static bool IsValidLineFont(int font);
static bool IsValidLineWeight(int weight);
static bool IsValidColor(int color);
static bool IsRegularPictureId(int id);
```

4.4.1 コンストラクタ

```
AwItemAttributes();
```
このオブジェクトはアイテムタイプ、ピクチャ番号が不正で、IsValid() メソッドが false を返します。

```
AwItemAttributes(int pid, int type,  int cls = 1, int rev = 1,  int font = 1, int weight = 1,
                 int color = 1, int blank = 0);
```
アイテム属性を渡すコンストラクタ。

4.4.2 アクセッサ

それぞれの属性ごとに値取得／設定メソッドがあります。設定メソッドは引数が正しい値の場合だけ更新します。

アイテムタイプを取得／設定する。
```
int  GetType() const;
void SetType(int type);
```

アイテムのクラス番号を取得／設定する。
```
int  GetClass() const;
void SetClass(int cls);
```

アイテムのレビジョン番号を取得／設定する。
```
int  GetRevision() const;
void SetRevision(int rev);
```

アイテムの線種番号を取得／設定する。
```
int  GetLineFont() const;
void SetLineFont(int font);
```

アイテムの線幅番号を取得／設定する。
```
int  GetLineWeight() const;
void SetLineWeight(int weight);
```

アイテムの色番号を取得／設定する。
```
int  GetColor() const;
void SetColor(int color);
```

アイテムのブランキングフラグを取得／設定する。
```
bool IsBlanked() const;
void SetBlanked(bool blank);
```

アイテムのピクチャ番号を取得／設定する。
```
int  GetPictureId() const;
void SetPictureId(int pid);
```

4.4.3 メソッド

```
bool IsValid() const;
```
アイテム属性の全ての値が適切な値であるとき true を返します。

4.5 アイテム属性の現在値

新しくアイテムを作成するときに使用するアイテム属性を「現在値」と呼びます。ユーザはコマンドでアクティブピクチャの番号、アイテムのクラス番号、レビジョン番号、線種、線幅などの「現在値」を変更します。「アイテム属性の現在値」はスクリーンの一部にステータスとして表示されています。

AwItemAttributes2 クラスは「アイテム属性の現在値」を表現します。このクラスは AwItemAttributes クラスを継承しています。AwItemAttributes2 オブジェクトは AwModel オブジェクトから得ます。

AwItemAttributes2 オブジェクトの値を変更してもスクリーンのステータス表示は更新しません。オブジェクトの値を一時的に変更したなら、後で元の値に戻しておくのが安全です。
次に示すコードフラグメントはピクチャ番号を一時的に変更する例です。

```
AwItemAttributes2* pItemAttr = pModel->GetItemAttributes();
int actpic = pItemAttr->GetPictureId();  // ピクチャの現在値を保持しておく。
pItemAttr->SetPictureId(31);             // ピクチャ番号を 31 に変更する。
// ここで何かする。
pItemAttr->SetPictureId(actpic);         // ピクチャ番号を元に戻す。
```

このオブジェクトの値は、空のテンポラリアイテムを作成するときにアイテム属性を指定しないと、デフォルトのアイテム属性として使用されます。この後説明する AwTempItemDb クラスの OpenItem() メソッドがそうです。
テンポラリアイテム作成前に AwItemAttributes2 オブジェクトの値を変更しておくよりも、作成したテンポラリアイテムのアイテム属性を変更するの方がほとんどの場合簡単です。またテンポラリアイテムのアイテムタイプは必ず設定します。
AwItemAttributes2 オブジェクトの値を一時的に変更する方法はあまり勧めません。

4.6 AwTempItemDb クラス

このクラスはテンポラリアイテムデータベースを表現します。テンポラリアイテムデータベースはアイテムを作成しパーマネントアイテムデータベースに保存するのに使う一時的なデータベースです。テンポラリアイテムデータベースに作成するアイテムをテンポラリアイテムと呼びます。パーマネントアイテムを変更するには、一旦テンポラリアイテムを作成し、テンポラリアイテムを変更し、パーマネントアイテムデータベースを更新します。
AwTempItemDb オブジェクトは、テンポラリアイテムが作成、変更、削除されても、その結果をスクリーンに反映させることはありません。テンポラリスクリーンを再表示する関数がテンポラリアイテムをスクリーンに表示します。
AwTempItemDb オブジェクトは AwModel オブジェクトから得ます。

4.6.1 アイテム数の上限

テンポラリアイテム数には上限があります。この値はアプリケーション起動時に一度だけ設定されます。その後は変更してはなりません。この値は全ての AwTempItemDb オブジェクトに共通です。

```
static int GetMaxItemCount();
```
テンポラリアイテム数の上限値を返します。

4.6.2 アイテム数

テンポラリアイテム数の上限を超えてテンポラリアイテムを作成しようとすると、全てのテンポラリアイテムをパーマネントアイテムデータベースに保存し、クリアしてから、新しいテンポラリアイテムを作成します。テンポラリアイテムの ID は 1 から始まる番号で指定します。

このオブジェクトが持つアイテム数を調べるメソッドがあります。

```
int GetItemCount() const;
```
テンポラリアイテムの数を返します。
```
bool IsEmpty() const;
```
テンポラリアイテムがひとつもないとき true を返します。
```
bool IsFull() const;
```
テンポラリアイテム数が上限に達しているとき true を返します。

次のメソッドはテンポラリアイテムデータベースを初期化します。全てのテンポラリアイテムが削除されます。
```
void Init();
```

4.6.3 新アイテム

ここで説明するメソッドは新しい空のテンポラリアイテムを作成します。テンポラリアイテムにはアイテム属性だけが設定されます。パーマネントアイテムをコピイしてテンポラリアイテムを作るには後で説明する LoadItem() メソッドを使います。

次のメソッドは空のテンポラリアイテムを作成します。アイテム属性は「アイテム属性の現在値」を設定します。この後にアイテムタイプを設定しなければなりません。
```
AwTempItem* OpenItem();
```
戻り値　AwTempItem オブジェクトへのポインタを返します。失敗したときは null ポインタを返します。

次のコードフラグメントは空のテンポラリアイテムを作成し、アイテムタイプを設定します。
```
AwTempItem* pTempItem = pModel->GetTempItemDb()->OpenItem();
if (pTempItem == NULL)
  return;
pTempItem->SetType(AwItem::POINT);
```

次のメソッドも空のテンポラリアイテムを作成します。このメソッドは引数を使ってアイテム属性をコントロールできます。また更新と複製の選択ができます。このテンポラリアイテムをパーマネントアイテムデータベースに保存するとき、元のパーマネントアイテムを更新するか、新しいアイテムとして追加するかということです。
```
AwTempItem* OpenItem(int idptr, int keydup, int keypic);
```
idptr　アイテム識別子。
- 0　新しいアイテムを作成する。アイテム属性は「アイテム属性の現在値」を設定します。
- 正の値　アイテム識別子。このアイテムが存在しなければエラーです。このアイテムから取得したアイテム属性は keydup に従って設定します。

keydup　更新／複製を指示します。1 <= idptr の場合のみ有効です。
- 0　更新。アイテム属性は元のアイテムと同じに設定します。このテンポラリアイテムをパーマネントアイテムデータベースに保存すると元のアイテムを更新します。
- 1　複製。アイテム属性は元のアイテムと同じに設定します。このテンポラリアイテムをパーマネントアイテムデータベースに保存すると新しいアイテムとして追加し元のアイテムは変更しません。
- 2　複製。アイテム属性（クラス、レビジョン、線種、線幅）には「現在値」を設定します。このテンポラリアイテムをパーマネントアイテムデータベースに保存すると新しいアイテムとして追加し元のアイテムは変更しません。

keypic　テンポラリアイテムのピクチャ番号。
- 0　上記の規則で設定したピクチャ番号を使います。
- 正の値　上記の規則で設定したピクチャ番号ではなくこのピクチャ番号を使います。

戻り値　AwTempItem オブジェクトへのポインタを返します。失敗したときは null ポインタを返します。

4.6.4 アイテムロード

パーマネントアイテムをコピイしてテンポラリアイテムを作成します。更新と複製の選択ができます。このテンポラリアイテムをパーマネントアイテムデータベースに保存するとき、元のパーマネントアイテムを更新するか、新しいアイテムとして追加するかということです。
アイテム属性は元のアイテムと同じに設定します。元のアイテムの持つサブレコードもコピイします。複製の場合はアソシエーションサブレコード（AwSr::ASSOCIATION）はコピイしません。最後に AwSr::EOI サブレコードが付いていることに注意してください。EOI サブレコードの後ろにサブレコードを追加しないようにしなければなりません。

```
AwTempItem* LoadItem(int key, const AwItem& item);
```

key　更新／複製を指示します。
- 1　更新。このテンポラリアイテムをパーマネントアイテムデータベースに保存すると元のアイテムを更新します。
- 2　複製。このテンポラリアイテムをパーマネントアイテムデータベースに保存すると新しいアイテムとして追加し元のアイテムは変更しません。

item　パーマネントアイテム。
戻り値　AwTempItem オブジェクトへのポインタを返します。失敗したときは null ポインタを返します。

4.6.5 アイテムの取得

テンポラリアイテムを得ます。テンポラリアイテムの ID は作成順に 1 から始まる番号で指定します。

```
AwTempItem* GetItem(int id);
```

id　テンポラリアイテム ID（1 <= id <= GetItemCount()）。
戻り値　AwTempItem オブジェクトへのポインタを返します。アイテムがなければ null ポインタを返します。

最後のテンポラリアイテム（テンポラリアイテム ID が最大）を得ます。

```
AwTempItem* GetLastItem();
```

戻り値　AwTempItem オブジェクトへのポインタを返します。アイテムがなければ null ポインタを返します。

4.6.6 アイテムの削除

テンポラリアイテムアイテムを削除します。アイテムを削除すると、削除されたアイテムの後方にあるアイテムの ID は先頭から数えた番号（1 から始まる整数）に変わります。

指定したテンポラリアイテムを削除します。

```
void RemoveItem(int id);
```

id　テンポラリアイテム ID（1 <= id <= GetItemCount()）。

最後のテンポラリアイテム（テンポラリアイテム ID が最大）を削除します。

```
bool RemoveLastItem(G2Point* pnt = NULL);
```

pnt　曲線アイテムの始点座標を得ることができます。アイテムに AwSr::STARTPOINT サブレコードがければ何もしません。この値が不要であれば引数を省略するか、null ポインタを渡します。
戻り値　アイテムを削除したとき true を返します。

複数のテンポラリアイテムを削除します。

```
bool RemoveItems(int fromId, int toId = MAX_ITEM_COUNT);
```

fromId　削除する最初のテンポラリアイテム ID（1 <= fromId <= GetItemCount()）。

toId　削除する最後のテンポラリアイテム ID（fromId <= toId <= GetItemCount()）。この引数を省略したときは、fromId を含めて以降の全てのアイテムを削除します。
戻り値　アイテムを削除したとき true を返します。

4.6.7　アイテムを保存

ここで説明するメソッドはテンポラリアイテムをパーマネントアイテムデータベースに保存します。テンポラリアイテム保存時にエラーが発生した場合は処理を中断します。エラーを起こしたテンポラリアイテム以降にあるテンポラリアイテムは処理しませんので注意してください。
最後にテンポラリアイテムアイテムを削除します。

このメソッドは唯一スクリーンの更新を行うメソッドです。パーマネントアイテムを更新する場合は、更新前にパーマネントアイテムをスクリーンからイレースします。そしてパーマネントアイテム更新後にスクリーンに表示します。先にイレースしておかないと、アイテムの更新前と更新後の表示が重なり不都合だからです。これは、更新または新たに追加したパーマネントアイテムだけに限定してスクリーン更新をするためです。

```
bool StoreItems(int repaint = 0, bool keepLast = false);
```
repaint　表示制御。
- 0　パーマネントアイテムを表示し、テンポラリスクリーンも再表示する。
- 1　パーマネントアイテムを表示する。
- 2　パーマネントアイテムをイレースするが表示しない。
- 3　何もしない。

keepLast 最後のテンポラリアイテムの制御。
- false　最後のテンポラリアイテムを含め全てのテンポラリアイテムを保存する。
- true　最後のテンポラリアイテムは保存しないでテンポラリアイテムのままにしておく。

戻り値　成功したとき true を返す。

4.7　AwTempItem クラス

このクラスはテンポラリアイテムを表現します。
AwTempItem オブジェクトは AwTempItemDb オブジェクトに作成します。

4.7.1　継承したメソッド

このクラスは抽象クラス AwItem を継承していますから、AwItem のメソッドを全て使用できます。下記のメソッドはパーマネントアイテムとは違っていますので注意してください。

```
int GetItemId() const;
```
テンポラリアイテムがパーマネントアイテム更新モードならパーマネントアイテムのアイテム識別子を返します。それ以外は 0 を返します。更新モードになるのは、OpenItem(idptr, 0, keypic) または LoadItem(1, item) です。

```
const std::string GetName() const;
```
テンポラリアイテムはアイテム名を持ちませんので常に空文字列を返します。

4.7.2　アイテム属性の変更

テンポラリアイテム作成メソッドがアイテム属性を設定します。その後、テンポラリアイテムのアイテム属性を変更するには下記のメソッドを使います。
それぞれの属性ごとに設定メソッドがあります。

```
void SetType(int newval);
void SetClass(int newval);
```

```
void SetRevision(int newval);
void SetLineFont(int newval);
void SetLineWeight(int newval);
void SetColor(int newval)
void SetBlanked(bool newval)
void SetPictureId(int newval);
```

アイテム属性を一括変更するメソッドがあります。
AwItemAttributes オブジェクトを使ってこのアイテムのアイテム属性を設定します。

```
void SetAttributes(const AwItemAttributes& attr);
```

attr　AwItemAttributes オブジェクト。

AwSrAttributes オブジェクトを使ってこのアイテムのアイテム属性を設定します。AwSrAttributes オブジェクトはピクチャ番号を持ちませんのでピクチャ番号は変更できません。

```
void SetAttributes(const AwSrAttributes& srAttr);
```

srAttttr　AwSrAttributes オブジェクト。

4.7.3　サブレコードの追加、削除

サブレコードはインデックス（0 から始まる）を指定して取得します。サブレコードは任意の順序でアクセスできます。テンポラリアイテムには non-const の GetSrAt() メソッドがあります。このメソッドは は non-const ポインタを返すので、オブジェクトのメソッドを使って内容を変更することができます。

```
AwSr* GetSrAt(int index);
```

index　サブレコードの位置を示すインデックス。0 <= index < GetSrCount()。
戻り値　index の位置の AwSr オブジェクトへのポインタを返します。index が範囲外のときは null ポインタを返します。

テンポラリアイテムにはサブレコードの追加、挿入、削除ができます。アイテムが AwSr::EOI サブレコードを持つ場合、その後ろにサブレコードを追加／挿入してはなりません。

アイテムの最後にサブレコードを追加します。

```
bool AddSr(const AwSr& sr);
```

sr　サブレコード。
戻り値　アイテムにサブレコードを追加したら true を返します。

アイテムにサブレコードを挿入します。インデックスで示された位置とそれに以降にあるサブレコードを後方に移動し、インデックスの位置にサブレコードを追加します。インデックス 0 はサブレコードを先頭に挿入します。

```
bool InsertSrAt(int index, const AwSr& sr);
```

index　サブレコードの挿入位置を示すインデックス。0 <= index <= GetSrCount()。
sr　サブレコード。
戻り値　アイテムにサブレコードを挿入したら true を返します。

インデックスの位置にあるサブレコードを新しいサブレコードで置き換えます。

```
bool SetSrAt(int index, const AwSr& sr);
```

index　サブレコードの位置を示すインデックス。0 <= index < GetSrCount()。
sr　サブレコード。
戻り値　アイテムにサブレコードを更新したら true を返します。

サブレコードを削除します。

```
bool RemoveSrAt(int from, int count = 1);
```

from　削除する先頭のサブレコードの位置を示すインデックス。0 <= index < GetSrCount()。
count　削除するサブレコード数。省略すると 1。インデックス from の位置から count で指定した数のサブレコードを削除します。
戻り値　サブレコードを削除したら true を返します。

全てのサブレコードを削除します。
```
void RemoveAllSrs();
```

次のメソッドはテンポラリアイテムをクローズします。これはアイテムの最後に AwSr::EOI サブレコードを追加します。Close() メソッドを使った後でもサブレコードの追加／挿入／設定／削除のメソッドを使えますが、AwSr::EOI サブレコードの後にサブレコードを追加してはなりません。それは不正なアイテムと判定します。
```
bool Close();
```
戻り値　アイテムの最後に AwSr::EOI サブレコードを追加したら true を返します。既にこのアイテムの最後のサブレコードが AwSr::EOI であれば何もせず true を返します。

アイテムの最後を示す AwSr::EOI サブレコードを削除します。
```
bool RemoveEoi();
```
戻り値　AwSr::EOI サブレコードを削除したら true を返します。

4.7.4　アイテムの検査

アイテムオブジェクトのデータが正しいか判定します。
(1)　**アイテム属性の値が全て正しいか。**
(2)　**最後のサブレコードは AwSr::EOI になっているか。**
(3)　**AwSr::EOI の前にひとつ以上サブレコードがあること。**
(4)　**サブレコード数が上限値を超えていないか。**
(5)　**サブレコード合計データサイズが上限値を超えていないか。**

```
int IsValid(bool bData = false) const;
```
bData　true を設定すると全てのサブレコードの内容検査も行います。
戻り値　このアイテムは有効と判断すると 0 を返します。

4.8　AwItemIdArray クラス

このクラスはアイテム識別子の可変長配列です。この配列は最初は空で、アイテム識別子を追加するとサイズを拡張していきます。配列の最大サイズに達するとそれ以上追加できなくなります。
アイテム識別子は正の整数です。0 はアイテム識別子ではないので、この配列には 0 を追加、設定することはできません。アクティブリストは特別で負のアイテム識別子を扱います。これは曲線アイテムが進行方向に対して反転していることを表します。アイテム識別子の比較では符号を無視します。
このクラスはデフォルトではアイテム識別子の重複検査を行いません。アイテム識別子を追加するときに重複検査をするようにもできます。

4.8.1　コンストラクタ

```
AwItemIdArray(size_t size = 2048);
```
配列の最大サイズ size を指定します。

4.8.2　条件設定

この配列が保持するアイテム識別子の数の上限を設定します。アイテム識別子を追加した後でもこのメソッドを使って上限を変更できます。ただし実際に格納されているアイテム識別子数はかわりません。実際に格納されているアイテム識別子数より小さな数を設定してもアイテム識別子数は減りません。
```
void SetMaxCount(size_t size);
```
size　　配列の最大サイズ。

アイテム識別子の重複検査を設定します。デフォルトは重複検査なしです。重複検査なしの場合は利用者側で重複するアイテム識別子を追加しないようにしなければなりません。

void SetUniqueId(bool unique)
　unique　　重複検査を有効にする (true)、無効にする (false)。

4.8.3　アイテム識別子の取得

配列が空か判定します。
```
bool IsEmpty() const;
```
　この配列が空のとき true を返します。

配列のアイテム識別子の数が上限に達しているか判定します。
```
bool IsFull() const;
```
　戻り値　この配列のアイテム識別子数が上限以上のとき true を返します。

配列のアイテム識別子の数を得ます。
```
int GetCount() const;
```
　戻り値　この配列のアイテム識別子の数を返します。

配列に追加できるアイテム識別子の数を得ます。
```
int GetRemainder() const;
```
　戻り値　この配列に追加できるアイテム識別子の数を返します。

配列の指定した位置のアイテム識別子を得ます。
```
int GetAt(int index) const;
```
　index　　アイテム識別子の位置を示すインデックス。0 <= index < GetCount()。
　戻り値　index の位置のアイテム識別子を返します。インデックスが範囲外のときは 0 を返します。

配列の最後のアイテム識別子を得ます。
```
int GetLast() const;
```
　戻り値　最後のアイテム識別子を返します。配列が空の時はのときは 0 を返します。

アイテム識別子の int 配列への const ポインタを得ます。
```
const int* GetPointer() const;
```
　戻り値　int 配列へのポインタを返します。配列が空のときは null ポインタを返します。
このメソッドは関数の引数がアイテム識別子の int 配列の場合に使うためにあります。
```
  Rptitems(1, 0, temids.GetPointer(), itemids.GetCount());
```

4.8.4　アイテム識別子の追加、削除

Add(), Append(), SetAt() メソッドはアイテム識別子の重複検査が有効なら重複検査を行います。

アイテム識別子をこの配列の最後に追加します。
```
bool Add(int id);
```
　idptr　　アイテム識別子。
　戻り値　追加したとき true を返します。

複数のアイテム識別子をこの配列の最後に追加します。
```
bool Append(const AwItemIdArray& itemIds, int first = 0, int last = INT_MAX);
```
　itemIds　AwItemIdArray オブジェクト。
　first　　追加するアイテム識別子の開始インデックス (0 <= first < itemIds.GetCount())。
　last　　追加するアイテム識別子の終了インデックス (first <= last < itemIds.GetCount())。省略または、itemIds.GetCount() <= last のときは最後まで。
　戻り値　全部追加できたとき true を返します。部分追加したときや何も追加しなかったら false を返します。

複数のアイテム識別子をこの配列の最後に追加します。
```
bool Append(int count, const int ids[]);
```
count　配列 ids 内のアイテム識別子の数。
ids　アイテム識別子を格納した配列。
戻り値　全部追加できたとき true を返します。部分追加したときや何も追加しなかったら false を返します。

配列の指定した位置のアイテム識別子の値を変更します。
```
bool SetAt(int index, int id);
```
index　位置を示すインデックス。0 <= index < GetCount()。
id　アイテム識別子。
戻り値　値を変更したとき true を返します。

配列を空にします。
```
void RemoveAll();
```

配列の最後のアイテム識別子を削除します。
```
bool RemoveLast();
```
戻り値　この配列が空なら false、削除したとき true を返します。

配列の指定した位置のアイテム識別子を削除します。
インデックス index の位置から count で指定した数のアイテム識別子を削除します。
```
void RemoveAt(int index, int count = 1);
```
index　削除する位置を示すインデックス。0 <= index < GetCount()。
count　削除する数。index < index + count <= GetCount()。

この配列から指定したアイテム識別子を削除します。
```
bool Remove(const AwItemIdArray& itemIds);
```
itemIds　AwItemIdArray オブジェクトが含むアイテム識別子を削除します。
戻り値　削除したとき true を返します。

この配列から指定したアイテム識別子を削除します。
```
bool Remove(int count, const int ids[]);
```
count　配列 ids 内のアイテム識別子の数。
ids　アイテム識別子を格納した配列。
戻り値　削除したとき true を返します。

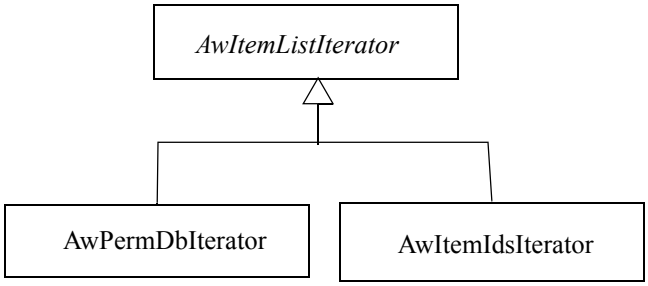
4.8.5　探索

この配列が指定したアイテム識別子を含むか調べます。
```
bool Contains(int id) const;
```
id　アイテム識別子。
戻り値　この配列がアイテム識別子を含むとき true を返します。

この配列が指定したアイテム識別子を含むか調べます。
```
int Find(int id) const;
```
id　アイテム識別子。
戻り値　この配列がアイテム識別子を含むとき、配列のインデックス (0 <= index < GetCount()) を返します。含まないときは -1 を返します。

4.9　データベースイテレータ

パーマネントアイテムをアクセスする手段としてイテレータを提供します。



4.9.1 AwItemListIterator インターフェイス

このクラスは下記の純粋仮想メソッドを定義する抽象クラスです。

```
bool HasNext();
```
次のアイテムがあるか調べます。次のアイテムがあるとき true を返します。

```
const AwItem* NextItem();
```
HasNext() が true を返したとき、パーマネントアイテムのポインタを得ます。次に HasNext() を呼び出すまでは同じ値を返します。

```
void Restart();
```
もう一度先頭からイテレートします。

このイテレータを使うには次のようにします。

```
while (iterator->HasNext()) {
  const AwItem* pItem = iterator->NextItem();
  // 省略
}
```

while ループの条件式の HasNext() メソッドが次のアイテムがあるかを調べます。メソッドは次のアイテムがあるとき true を返します。true のときは NextItem() メソッドで次のパーマネントアイテムのポインタを得ます。次に HasNext() メソッドが呼ばれるまで、NextItem() メソッドは同じものを返します。

これから説明する AwPermDbIterator クラス、AwItemIdsIterator クラスは AwItemListIterator インターフェイスを実装する具象クラスです。

4.9.2 AwPermDbIterator クラス

このクラスはパーマネントアイテムデータベースのアイテムを順次取り出すイテレータです。このイテレータでは探索対象ピクチャを指定し、アイテムタイプ、クラスなどで取り出すアイテムを絞り込みたいときにはアイテム選択マスクを使うことができます。
このイテレータは先頭のアイテム識別子から開始し最大アイテム識別子までを対象とします。このイテレータを使用している間にパーマネントアイテムが追加されても、それらは対象にはなりません。

4.9.2.1 コンストラクタ

```
AwPermDbIterator(const AwPermItemDb* pItemDb);
```

4.9.2.2 対象ピクチャ

探索対象ピクチャを設定する三つのメソッドがあります。そのどれかのメソッドを使って探索対象ピクチャを指定しなければなりません。デフォルトの探索対象ピクチャはありませんので、必ず設定しなければなりません。

探索対象ピクチャを設定する。
```
bool SelectPicture(int pid);
```
pid　　ピクチャ番号。

ピクチャ番号 fromPid から toPid までを探索対象ピクチャにする。
```
void SelectPictures(int fromPid, int toPid);
```
fromPid　先頭のピクチャ番号。
toPid　　最後のピクチャ番号。1 <= fromPid <= toPid。

探索対象ピクチャを配列で設定する。
```
void SelectPictures(int count, const int pids[]);
```
count　　配列内のピクチャ番号の数。
pids　　探索対象のピクチャ番号を格納した配列。

4.9.2.3　アイテム選択マスク

アイテム選択マスクを使うためのメソッドがあります。選択マスクはアイテムタイプ、クラスなどで取り出すアイテムを絞り込みたいときに使います。

アイテム選択マスクを有効／無効にします。デフォルトは無効です。
```
void SetMaskEnabled(bool enable);
```
enable　true はアイテム選択マスクを有効に、false は無効にします。

アイテム選択マスクオブジェクトを得ます。このオブジェクトのマスクを設定します。
```
AwSelectableMask* GetMask();
```

アイテム選択マスクを使うには次のようにします。
```
// アイテム選択マスクオブジェクトを得ます。
AwSelectableMask* pMask = iterator->GetMask();
// このオブジェクトのマスクを設定します。線分アイテムだけを取り出します。
pMask->SetSelectableAll(AwItemAttributes::BY_ITEMTYPE, false);
pMask->SetSelectable(AwItemAttributes::BY_ITEMTYPE, AwItem::LINE, true);
// アイテム選択マスクを有効にします。
iterator->SetMaskEnabled(true);
```

4.9.3　AwItemIdsIterator クラス

このクラスは与えられたアイテム識別子の配列を使ってアイテムを順次取り出すイテレータです。HasNext() メソッドはアイテム識別子の配列を先頭から順番にアクセスします。HasNext() メソッドはアイテム識別子で特定するアイテムが存在しないか、それが削除アイテムのときは、それを無視して次へ進みます。イテレータを使用している間はアイテム識別子の配列の内容を変更してはなりません。

次の三つのコンストラクタがあります。コンストラクタにアイテム識別子の配列を渡します。三番目のコンストラクタはアイテム識別子の配列ではなく、イテレートするアイテム識別子の範囲を与えます。

```
AwItemIdsIterator(const AwPermItemDb* pItemDb, const AwItemIdArray& itemIds);
```
アイテム識別子を持つ AwItemIdArray オブジェクトをイテレートします。イテレーション中は AwItemIdArray オブジェクトの内容を変更してはなりません。
```
AwItemIdsIterator(const AwPermItemDb* pItemDb, int count, const int itemIds[]);
```
アイテム識別子を持つ int 配列をイテレートします。イテレーション中は int 配列の内容を変更してはなりません。
```
AwItemIdsIterator(const AwPermItemDb* pItemDb, int firstid, int lastid);
```
イテレートする最初と最後のアイテム識別子を与えます。1 <= firstid <= lastid。

4.10 関数

● 関数一覧

関数名	機能
GetActiveModel	アクティブモデルを得る。
Dbdlitems	アクティブモデルのアイテムを削除する。
Dbdlitmpc	アクティブモデルの指定ピクチャのアイテムを削除する。
DbUundo	UNDO または UN-UNDO する。
Tmpgrptn	テンポラリアイテムを表示する。

4.10.1 GetActiveModel() 関数

アクティブモデルを得ます。
```
AwModel* GetActiveModel();
```
戻り値　AwModel オブジェクトへのポインタ。

4.10.2 Dbdlitems() 関数

アクティブモデルのアイテム削除する
```
int Dbdlitems(int keyud, const AwItemIdArray& itemids, bool erase = true)
```
keyud　1 とする。
itemids　アイテム識別子を持つ AwItemIdArray オブジェクトの参照。
erase　削除したアイテムは画面上から消去するとき true とする。
戻り値　0：正常終了。

```
int Dbdlitems(int key_ud, const int itmlst[], int nitms, bool erase = true)
```
keyud　1 とする。
itmlst　アイテム識別子の配列。
nitms　アイテム識別子の数。
erase　削除したアイテムは画面上から消去するとき true とする。
戻り値　0：正常終了。

4.10.3 Dbdlitmpc() 関数

アクティブモデルの指定ピクチャのアイテムを削除します。背景イメージは削除しません。削除されたアイテムは画面上から消去されます。
```
int Dbdlitmpc(int keypic, const AwSelectableMask* pSelectableMask);
```
keypic　ピクチャ制御スイッチ
0　：全てのピクチャのアイテムを削除する。
1 - AwItemAttributes::MAX_PICTURE:指定ピクチャのアイテムを削除する。
pSelectableMaskアイテム選択マスクオブジェクトへのポインタ
null　：アイテム選択マスクオブジェクトを指定しない。
! null　：アイテム選択マスクに問い合わせ選択可能なアイテムだけを削除する。
戻り値　0：正常終了。

4.10.4 DbUundo() 関数

UNDO または UN-UNDO する。
int DbUundo(int mode, int idptr)
mode　UNDO 実行モード
0　:UN-UNDO する。
1　: 最後の 1 つを UNDO する。
2　: 最後の UNDO ブロックを UNDO する。
3　: 指示したアイテムを UNDO する。
4　: 指示したアイテムを含む UNDO ブロックを UNDO する。
5　: 最後の削除 UNDO ブロックを UNDO する。
6　: 現在の UNDO ブロックを UNDO する。
idptr　アイテム識別子
UNDO 実行モード mode == 3, mode ==4 のときのみ参照。
戻り値　0：正常終了

4.10.5 Tmpgrptn() 関数

テンポラリアイテムを画面に表示／消去する。
void Tmpgrptn(int ids, int ide, int mode, int vtxswt)
ids　最初のテンポラリアイテムの番号。
ide　最後のテンポラリアイテムの番号。1 ≦ ids ≦ ide。
mode　表示 , 消去の指定。0 : 表示、1 : 消去。
vtxswt　自由曲線の制御点列の表示の指定。1 : 表示、0 : 表示しない。

第 5 章　アイテムのデータ構造

以下の表はアイテムタイプの一覧です。
アイテムタイプは整数番号ですが、それを表すアイテムタイプ定数も示します。アイテムタイプ定数はヘッダーファイル AwItem.h で定義しています。本書中の関数でアイテムタイプを指示するものがいくつかあります。その場合は、アイテムタイプ番号を記入するよりは、アイテムタイプ定数を記入する方が分かりやすいでしょう。

アイテムタイプ一覧表

番号	アイテム	タイプ定数	区分
1	点 (Point)	AwItem::POINT	図形
2	直線 (Line)	AwItem::LINE	図形
3	円／円弧 (Circular Arc)	AwItem::CIRCLE	図形
4	自由曲線 (Spline)	AwItem::SPLINE	図形
5	ストリングアイテム (String curve)	AwItem::STRING	図形
6	円錐曲線 (Conic curve - 予約)		図形
7-8	未使用		図形
9	複合アイテム (Composit item)	AwItem::COMPOSITE	構造化
10	未使用		
11	ジェネラルテキスト (General note, General label, Reference note, Reference label, Cutting plane line)	AwItem::TEXT	製図
12	マーク (General mark)	AwItem::MARK	製図
13	寸法 (Dimension)	AwItem::DIMENSION	製図
14	幾何公差 (Geometrical Tolerance)	AwItem::GTOL	製図
15	ハッチングアイテム (Hatching)	AwItem::XHT	製図
16	塗り潰しアイテム (Area fill)	AwItem::AFL	製図
17-20	未使用		製図
21-26	未使用		構造化
27	メンバアイテム	AwItem::MEMBER	構造化
28	APG アイテム	AwItem::APG	構造化

番号	アイテム	タイプ定数	区分
29	アソシエイトアイテム	AwItem::ASSOCIATION	構造化
30	シンボル (Symbol)	AwItem::SYMBOL	構造化
31	サブモデル (Submodel)	AwItem::SUBMODEL	構造化
32	イメージアイテム (V19-)	AwItem::IMAGE	SXF
33	SXF 作図部品定義 (V18-)	AwItem::SXFSFIGORG	SXF
34	SXF 作図部品配置 (V18-)	AwItem::SXFSFIGLOC	SXF
35	SXF 元図定義 (ハッチング領域) (V18-)	AwItem::SXFCRVDEF	SXF
36	SXF 元図定義 (楕円、文字) (V18-)	AwItem::SXFORGDE	SXF

アイテムのデータはサブレコードの並びで表現します。アイテムタイプにより、どのサブレコードをどの順序に並べるかが決まっています。アイテム属性のアイテムタイプとサブレコード構成とが合致しなくてはなりません。線分アイテムが円弧サブレコードを持つようなことは矛盾を生じます。
以下にアイテムタイプ毎のサブレコードの並びを記述します。以下のサブレコード構成は原則的なものです。アイテムにプロパティを追加する場合は、原則的なサブレコードの並びの後、アイテム終了サブレコード (AwSr::EOI) の前に追加します。プロパティを原則的なサブレコードの並びの間に挿入するのは誤りです。
サブレコードタイプは整数番号ですが、以下では、サブレコードタイプ定数で表記しています。サブレコードタイプ定数はヘッダーファイル AwSr.h で定義しています。

説明の中で「レコード」という言葉を使います。たとえば、ジェネラルラベルやリファレンスラベルのアイテムは引出し線を持ちます。この引出し線は共通のサブレコード構成で、どのサブレコードをどの順序に並べるかが決まっています。このようにアイテムの一部分で、特定のものを表現するサブレコードの並びをレコードと呼んでいます。通常、レコードの先頭にはカテゴリサブレコード (AwSr::CATEGORY) を持ちます。ひとつのレコードは、カテゴリサブレコード で始まり、次のカテゴリ サブレコードまたはアイテムの終了サブレコード (AwSr::EOI) の直前までです。カテゴリサブレコードによって、このレコードが何を表すかを区別します。カテゴリを表す定数はヘッダーファイル AwSrCategory.h で定義しています。

5.1 点アイテム

サブレコード構造
```
0.      Point (X,Y)(AwSr::POINT)
1.      End of item(AwSr::EOI)
```

アイテムの最小／最大座標値
```
Xmin = X
Ymin = Y
Xmax = X
Ymax = Y
```

5.2 線分アイテム

サブレコード構造
```
0.      Start point(Xs,Ys)(AwSr::STARTPOINT)
1.      Line(Xe,Ye)(AwSr::LINE)
2.      End of item(AwSr::EOI)
```

アイテムの最小／最大座標値
Xmin = MIN(Xs,Xe)
Ymin = MIN(Ys,Ye)
Xmax = MIN(Xs,Xe)
Ymax = MIN(Ys,Ye)

制約
部分線種／線幅の変更をほどこした線分アイテムは複数の線分に分割される。
このとき分割された線分すべてが一直線上に順番になければならない。また、先頭と最後の線分は非表示にはできない。非表示にできるのは両端以外である。このときのサブレコードの並びは、次のようになる。
AwSr::STARTPOINT, AwSr::LINE, …, AwSr::LINE, AwSr::EOI

5.3 円／円弧アイテム

サブレコード構造
0.Start point(Xs,Ys)(AwSr::STARTPOINT)
1.Arc(Xm,Ym,Xe,Ye,Xc,Yc,radius,angle)(AwSr::CIRCLE)
2.End of item(AwSr::EOI)

制約
円は円弧の特殊解 (始点 == 終点) として処理する。
部分線種／線幅の変更をほどこした円弧アイテムは複数の円弧に分割される。
このとき分割された円弧すべてがひとつの円上に順番になければならない。また、先頭と最後の円弧は非表示にはできない。非表示にににできるのは両端以外である。このときのサブレコードの並びは、つぎのようになる。
AwSr::STARTPOINT, AwSr::CIRCLE, …, AwSr::CIRCLE, AwSr::EOI

5.4 自由曲線アイテム

サブレコード構造
0. Start point(AwSr::STARTPOINT)
1. Bezier curve #1(AwSr::BEZIER)
2. Bezier curve #2(AwSr::BEZIER)
:
m. Bezier curve #m(AwSr::BEZIER)
m+1. End of item(AwSr::EOI)

ここで、m は３次 Bezier カーブセグメントの数。

アイテムの最小／最大座標値
Xmin = MIN（各サブレコードの最小 X)
Ymin = MIN（各サブレコードの最小 Y)
Xmax = MAX（各サブレコードの最大 X)
Ymax = MAX（各サブレコードの最大 Y)

補足
最初に入力点列を通過する Uniformed Non-rational B-Spline を計算します。次に各スパンごとに Cubic Bezier Curve に変換します。

5.5 ストリングアイテム

アイテムの定義

線分、円弧、３次 Bezier カーブセグメントの連なる曲線。
各セグメントは連続していて一筆書きできること。

サブレコード構造

0.	始点 (X1,Y1)(AwSr::STARTPOINT)
1.	線分、円弧 または ３次 Bezier(AwSr::LINE, AwSr::CIRCLE または AwSr::BEZIER)
:	
m.	線分、円弧 または ３次 Bezier(AwSr::LINE, AwSr::CIRCLE または AwSr::BEZIER)
m+1.	End of item(AwSr::EOI)

アイテムの最小／最大座標値
Xmin = MIN（各サブレコードの最小 X)
Ymin = MIN（各サブレコードの最小 Y)
Xmax = MAX（各サブレコードの最大 X)
Ymax = MAX（各サブレコードの最大 Y)

制約
部分線種変更をほどこす場合、先頭と最後の円弧は非表示にはできない。非表示ににできるのは両端以外である。

5.6 複合アイテム

アイテムの定義
いくつかのアイテムを集めて１つのアイテムとしたもの。

サブレコード構造

0.	もとのアイテムの属性(AwSr::ATTRIBUTES2)
1.	もとのアイテムのサブレコード
:	
m.	もとのアイテムの属性(AwSr::ATTRIBUTES2)
m+1.	もとのアイテムのサブレコード
:	
n.	End of item(AwSr::EOI)

制約
(1) もとのアイテムのサブレコードの並びがそのままの順序で取り込まれる。
ただし、アイテムの終りを示す AwSr::EOI だけは除かれる。
(2) 複合アイテムに複合アイテムに含めてもよいが、複合アイテムにはアイテムの階層はない。

アイテムの最小／最大座標値
Xmin = MIN（各サブレコードの最小 X)
Ymin = MIN（各サブレコードの最小 Y)
Xmax = MAX（各サブレコードの最大 X)
Ymax = MAX（各サブレコードの最大 Y)

5.7 グラフィックステキスト アイテム

アイテムの定義
ジェネラルノート／ジェネラルラベル
リファレンスノート／リファレンスラベル
切断線

レコード構造
ジェネラルノート、ジェネラルラベルはテキストと引出線の２種類のレコードで構成される。
ジェネラルノート

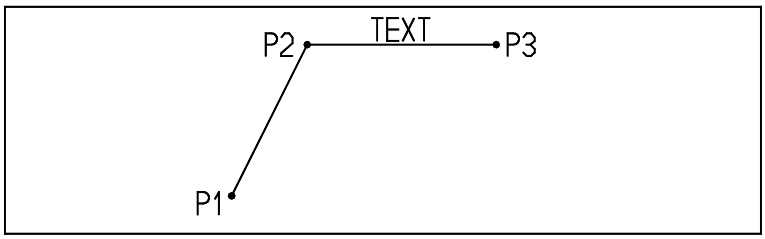
テキストレコード

ジェネラルラベル
テキストレコード
引出線レコード

リファレンスノート・リファレンスラベルは、テキスト・風船・引出線の３種類のレコードで構成される。
リファレンスノート
テキストレコード
風船レコード
リファレンスラベル
テキストレコード
風船レコード
引出線レコード

切断線はテキスト・矢印・切断線の３種類のレコードで構成される。テキスト、矢印はそれぞれ２つまで。
テキストレコード(×2)
矢印レコード　(×2)
切断線レコード

制約
(1) 引出線の点列は矢羽マークからテキストストリングへむかって並ぶ（下図のP1, P2, P3の順)。



(2) テキスト枠レコード、テキスト下線レコードが付くときは、テキストレコードの直後になければならない。

ジェネラルノート、ジェネラルラベル
(1) テキストレコード
0. カテゴリ (AwSr::CATEGORY)
メインカテゴリ AwSrCategory::TEXT
サブカテゴリ #0（未使用）
1. テキスト (AwSr::TEXT)

テキスト枠レコード（オプショナル）
0. カテゴリ (AwSr::CATEGORY)
メインカテゴリ AwSrCategory::TEXTBOX
サブカテゴリ #0（未使用）
1. 枠の始点 (AwSr::STARTPOINT)
2. 最初の線分 (AwSr::LINE)
3 2番目の線分 (AwSr::LINE)
4. 3番目の線分 (AwSr::LINE)
5. 最後の線分 (AwSr::LINE)
テキスト下線レコード（オプショナル）
0. カテゴリ (AwSr::CATEGORY)
メインカテゴリ AwSrCategory::UNDERLINE
サブカテゴリ #m（mは下線の数）

1. 下線の始点 (AwSr::STARTPOINT)
2. 下線の終点 (AwSr::LINE)
3. 下線の始点 (AwSr::STARTPOINT)
:
2m. 下線の終点 (AwSr::LINE)

(2) 引出線レコード
0. カテゴリ (AwSr::CATEGORY)
メインカテゴリ AwSrCategory::LEADER
サブカテゴリ #1
1. 引出線の始点 (AwSr::STARTPOINT)
2. 引出線の最初の線分 (AwSr::LINE)
:
m. 引出線の最後の線分 (AwSr::LINE)
m+1. 矢羽のマーク (AwSr::MARK)
m は引出線の点数。32 点以下。

リファレンスノート、リファレンスラベル
(1) テキストレコード
0. カテゴリ (AwSr::CATEGORY)
メインカテゴリ AwSrCategory::BALLOON
サブカテゴリ #0（文字列）
1. テキスト (AwSr::TEXT)

(2) 風船レコード
0. カテゴリ (AwSr::CATEGORY)
メインカテゴリ AwSrCategory::BALLOON
サブカテゴリ #1（マーク）
1. マーク (AwSr::MARK)

(3) 引出線レコード
0. カテゴリ (AwSr::CATEGORY)
メインカテゴリ AwSrCategory::LEADER
サブカテゴリ #1
1. 引出線の始点 (AwSr::STARTPOINT)
2. 引出線の最初の線分 (AwSr::LINE)
:
m. 引出線の最後の線分 (AwSr::LINE)
m+1. 矢羽のマークの原点 (AwSr::MARK)
m はリーダの点数。32 点以下。

切断線
(1) テキストレコード
0. カテゴリ (AwSr::CATEGORY)
メインカテゴリ AwSrCategory::CPL_TEXT
サブカテゴリ #1, #2（１番目か２番目かを示す）
1. テキスト (AwSr::TEXT)

(2) 矢印レコード
0. カテゴリ (AwSr::CATEGORY)
メインカテゴリ AwSrCategory::CPL_MARK
サブカテゴリ #1, #2（１番目か２番目かを示す）
1. 矢印マーク (AwSr::MARK)

(3) 切断線レコード
0. カテゴリ (AwSr::CATEGORY)
メインカテゴリ AwSrCategory::LEADER
サブカテゴリ #1
1. 切断線の始点 (AwSr::STARTPOINT)
2. 切断線の最初の線分 (AwSr::LINE)
:

m.　　切断線の最後の線分　　(AwSr::LINE)
m は切断線の点数。32 点以下

5.8　マークアイテム

アイテムの定義
　単純なマークだけのアイテム
　溶接記号
　面の肌の記号

単純なマークだけのアイテム サブレコード構造
　0.　マーク　　(AwSr::MARK)
　1.　End of item　　(AwSr::EOI)

溶接記号 レコード構造
　(1)　溶接記号レコード（矢の手前側）
　　0.　カテゴリ　　(AwSr::CATEGORY)
　　　メインカテゴリ　AwSrCategory::WELD
　　　サブカテゴリ　　#1
　　1.　マーク　　(AwSr::MARK)

　(2)　溶接の仕様レコード（矢の手前側）
　　このレコードは必要に応じて付ける。サブカテゴリの番号によって溶接部断面寸法やルート開先などの区別がなされる。したがって同じサブカテゴリのレコードが２つ以上あってはならない。
　　0.　カテゴリ　　(AwSr::CATEGORY)
　　　メインカテゴリ　AwSrCategory::WELD
　　　サブカテゴリ　　#11 ～ #13
　　1.　テキスト　　(AwSr::TEXT)

　(3)　溶接部の表面形状マークと仕上げ方法（矢の手前側）
　　このレコードは、必要に応じてひとつだけ付けることができる。
　　表面形状マーク
　　0.　カテゴリ　　(AwSr::CATEGORY)
　　　メインカテゴリ　AwSrCategory::WELD
　　　サブカテゴリ　　#4
　　1.　マーク　　(AwSr::MARK)
　　仕上げ方法の文字
　　0.　カテゴリ　　(AwSr::CATEGORY)
　　　メインカテゴリ　AwSrCategory::WELD
　　　サブカテゴリ　　#14
　　1.　テキスト　　(AwSr::TEXT)

　(4)　溶接記号レコード（矢の向う側）
　　0.　カテゴリ　　(AwSr::CATEGORY)
　　　メインカテゴリ　AwSrCategory::WELD
　　　サブカテゴリ　　#2
　　1.　マーク　　(AwSr::MARK)

　(5)　溶接の仕様レコード（矢の向う側）
　　このレコードは必要に応じて付ける。サブカテゴリの番号によって溶接部断面寸法やルート開先などの区別がなされる。したがって同じサブカテゴリのレコードが２つ以上あってはならない。
　　0.　カテゴリ　　(AwSr::CATEGORY)
　　　メインカテゴリ　AwSrCategory::WELD
　　　サブカテゴリ　　#21 ～ #23
　　1.　テキスト　　(AwSr::TEXT)

　(6)　溶接部の表面形状マークと仕上げ方法（矢の向う側）
　　このレコードは、必要に応じてひとつだけ付けることができる。
　　表面形状マーク

0. カテゴリ (AwSr::CATEGORY)
メインカテゴリ AwSrCategory::WELD
サブカテゴリ #5
1. マーク (AwSr::MARK)
仕上げ方法の文字
0. カテゴリ (AwSr::CATEGORY)
メインカテゴリ AwSrCategory::WELD
サブカテゴリ #24
1. テキスト (AwSr::TEXT)

(7) 現場溶接記号レコード（オプショナル）
0. カテゴリ (AwSr::CATEGORY)
メインカテゴリ AwSrCategory::WELD
サブカテゴリ #3
1. マーク (AwSr::MARK)

(8) 特記事項レコード（オプショナル）
0. カテゴリ (AwSr::CATEGORY)
メインカテゴリ AwSrCategory::WELD
サブカテゴリ #31
1. テキスト (AwSr::TEXT)

(9) 引出線レコード
0. カテゴリ (AwSr::CATEGORY)
メインカテゴリ AwSrCategory::LEADER
サブカテゴリ #1
1. 引出線の始点 (AwSr::STARTPOINT)
2. 引出線の最初の線分 (AwSr::LINE)
:
m. 引出線の最後の線分 (AwSr::LINE)
m+1. 矢羽のマーク (AwSr::MARK)
m は引出線の点数。32 点以下。

(10) 矢の向こう側を示す基線 (オプショナル)
0. カテゴリ (AwSr::CATEGORY)
メインカテゴリ AwSrCategory::LEADER
サブカテゴリ #2
1. 基線のパラメータ (AwSr::SCALAR)
基線の線種
基線間の距離
2. 基線の始点 (AwSr::STARTPOINT)
3. 基線の終点 (AwSr::LINE)

面の肌の記号 レコード構造

(1) 面の肌の記号レコード
0. カテゴリ (AwSr::CATEGORY)
メインカテゴリ AwSrCategory::TEXTURE
サブカテゴリ #1
1. マーク (AwSr::MARK)
面の肌に加工方法が付くとき、面の肌の記号に水平線が付く。
2. マークに付く線の始点 (AwSr::STARTPOINT)
3. マークに付く線の終点 (AwSr::LINE)

(2) 面の肌の仕様レコード
このレコードは必要に応じて付ける。サブカテゴリの番号によって平均粗さ、カットオフ値などの区別がなされる。したがって同じサブカテゴリのレコードが２つ以上あってはならない。
0. カテゴリ (AwSr::CATEGORY)
メインカテゴリ AwSrCategory::TEXTURE
サブカテゴリ #11 ～ #21

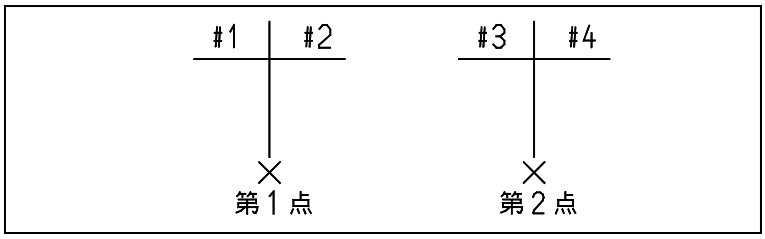
1. テキスト (AwSr::TEXT)

(3) 引出線レコード（オプショナル）
0. カテゴリ (AwSr::CATEGORY)
メインカテゴリ AwSrCategory::LEADER
サブカテゴリ #1
1. 引出線の始点 (AwSr::STARTPOINT)
2. 引出線の最初の線分 (AwSr::LINE)
:
m. 引出線の最後の線分 (AwSr::LINE)
m+1. 矢羽のマーク (AwSr::MARK)
m は引出線の点数。32 点以下。

5.9 寸法アイテム

レコードの説明

寸法線レコード
0. カテゴリ サブレコード (AwSr::CATEGORY)
メインカテゴリ AwSrCategory::DIM_ARROW
サブカテゴリ #1 ～ #4
サブカテゴリは寸法線の番号を表わす。



1. 寸法線始点 (AwSr::STARTPOINT)
2. 寸法線終点または円弧 (AwSr::LINE または AwSr::CIRCLE)
3. 寸法線の矢羽 (AwSr::MARK)

寸法線の矢羽は、なくてもよい。
各寸法線は一直線上にあること。
角度寸法，円弧長寸法のときは #2,#3 の寸法線は円弧になる。

寸法補助線レコード
0. カテゴリ サブレコード (AwSr::CATEGORY)
メインカテゴリ AwSrCategory::DIM_WITNESS
サブカテゴリ #1, #2 １番目か２番目かを示す。
1. 寸法補助線 始点 (AwSr::STARTPOINT)
2. 寸法補助線 終点 (AwSr::LINE)

寸法値 レコード
0. カテゴリ サブレコード (AwSr::CATEGORY)
メインカテゴリ AwSrCategory::DIM_TEXT
サブカテゴリ #0（寸法値）
1. 寸法値 (AwSr::TEXT)
64 バイト以内

公差値レコード
0. カテゴリサブレコード (AwSr::CATEGORY)
メインカテゴリ AwSrCategory::TOLERANCE

許容差種類　　#1　　± の寸法許容差
　　#2　　上の寸法許容差
　　#3　　下の寸法許容差

1. 公差値　　(AwSr::TEXT)
　32 バイト以内

± の寸法許容差を持つときは、上の寸法許容差・下の寸法許容差は持てない。
逆に、上の寸法許容差または下の寸法許容差を持つとき、± の寸法許容差は持てない。

付加文字列レコード

0. カテゴリサブレコード　　(AwSr::CATEGORY)
　メインカテゴリ AwSrCategory::DIM_TEXT
　サブカテゴリ　#1（付加文字列）
1. 付加文字列　　(AwSr::TEXT)
　64 バイト以内

引出線レコード

0. カテゴリ サブレコード　　(AwSr::CATEGORY)
　メインカテゴリ　AwSrCategory::LEADER
　サブカテゴリ　　#1
1. 引出線始点　　(AwSr::STARTPOINT)
2. 引出線の点　　(AwSr::LINE
　：
m. 引出線の点　　(AwSr::LINE)
m+1. 矢羽マーク　　(AwSr::MARK)

m は引出線の点数。32 点以下。

長さ寸法 (Linear Dimension)

レコード構造

0. 寸法パラメータ　　(AwSr::DIMENSION)
1. 第１寸法線レコード（オプショナル）
2. 第２寸法線レコード（オプショナル）
3. 第３寸法線レコード（オプショナル）
4. 第４寸法線レコード（オプショナル）
5. 第１寸法補助線レコード（オプショナル）
6. 第２寸法補助線レコード（オプショナル）
7. 寸法値 レコード
8. 公差値 レコード（オプショナル）
9. 付加文字列レコード（オプショナル）
10. End of Item　　(AwSr::EOI)

オプショナルのレコードはなくてもよい。

注意

レコードについては前記を参照のこと。
寸法補助線は互いに平行であること。

片寄せ寸法

第２寸法線レコードと第１寸法補助線レコードまたは
第１寸法線レコードと第１寸法補助線レコードを使用。

オーディネイト寸法

第２寸法補助線レコードだけを使用。

累進寸法

第３寸法線レコードと第２寸法線レコードだけを使用。

角度寸法 (Angular dimension)

レコード構造

```
0.  寸法パラメータ                    (AwSr::DIMENSION)
1.  第１寸法線レコード（オプショナル）
2.  第２寸法線レコード（オプショナル）
3.  第３寸法線レコード（オプショナル）
4.  第４寸法線レコード（オプショナル）
5.  第１寸法補助線 レコード（オプショナル）
6.  第２寸法補助線 レコード（オプショナル）
7.  寸法値 レコード
8.  公差値 レコード（オプショナル）
9.  付加文字列レコード（オプショナル）
10. End of Item                       (AwSr::EOI)
```

オプショナルのレコードはなくてもよい。

注意

レコードについては前記を参照のこと。

半径寸法 (Radius dimension)

レコード構造

```
0.  寸法パラメータ                    (AwSr::DIMENSION)
1.  引出線レコード
2.  反対側の引出線レコード（オプショナル）
    反対側の引出線は、寸法値テキストが付く引出線の矢羽の反対側に付ける線。
     0.  カテゴリサブレコード         (AwSr::CATEGORY)
           メインカテゴリ  AwSrCategory::RAD_LEADER
           サブカテゴリ    #0
     1.  引出線の始点                 (AwSr::STARTPOINT)
     2.  引出線の終点                 (AwSr::LINE)
3.  寸法値レコード
4.  公差値レコード（オプショナル）
5.  付加文字列レコード（オプショナル）
6.  End of item                       (AwSr::EOI)
```

注意

レコードについては前記を参照のこと。

直径寸法 (diameter dimension)

レコード構造

```
0.  寸法パラメータ                    (AwSr::DIMENSION)
1.  第１寸法線レコード（オプショナル）
2.  第２寸法線レコード（オプショナル）
3.  第３寸法線レコード（オプショナル）
4.  寸法値 レコード
5.  公差値 レコード（オプショナル）
6.  付加文字列レコード（オプショナル）
7.  End of Item                       (AwSr::EOI)
```

オプショナルのレコードはなくてもよい。

注意

レコードについては前記を参照のこと。
寸法テキストが円の外側に出るとき、外側に出る寸法線は常に第 1 寸法線レコードとする。

円弧長寸法 (Arclength dimension)

レコード構造
0. 寸法パラメータ (AwSr::DIMENSION)
1. 第１寸法線レコード（オプショナル）
2. 第２寸法線レコード（オプショナル）
3. 第３寸法線レコード（オプショナル）
4. 第４寸法線レコード（オプショナル）
5. 第１寸法補助線レコード（オプショナル）
6. 第２寸法補助線レコード（オプショナル）
7. 寸法値 レコード
8. 公差値 レコード（オプショナル）
9. 付加文字列レコード（オプショナル）
10. End of Item (AwSr::EOI)

オプショナルのレコードはなくてもよい。

チャンファ寸法 (Chamfer dimension)

レコード構造
0. 寸法パラメータ (AwSr::DIMENSION)
1. 第１寸法補助線（オプショナル）
2. 引出線レコード
3. 寸法値レコード
4. 公差値レコード（オプショナル）
5. 付加文字列レコード（オプショナル）
6. End of Item (AwSr::EOI)

オプショナルのレコードはなくてもよい。

注意
第 1 寸法補助線は、引出線の矢羽が面取り線の外に出るときに付ける。

5.10 幾何公差アイテム

レコード構造
0. カテゴリ サブレコード(AwSr::CATEGORY)
メイン カテゴリ AwSrCategory::GT
サブ カテゴリ 幾何公差データ行数 (m)
1. 幾何公差パラメータ(AwSr::GTFRAME)
3.幾何公差データレコード
幾何公差データ 1 行につき、以下のレコードがある。
幾何公差記号レコード
精度テキストレコード
データム文字レコード
幾何公差記入枠線レコード（複数）
2. 引出線レコード（オプショナル、2 つまで）

レコードの説明
幾何公差記号レコード
0. カテゴリレコード (AwSr::CATEGORY)
メインカテゴリ AwSrCategory::GT_MARK

```
        サブカテゴリ    行番号 (1 - m)
    1.  幾何公差マーク                  (AwSr::MARK)
```

精度テキストレコード

```
    0.  カテゴリレコード                (AwSr::CATEGORY)
        メインカテゴリ  AwSrCategory::GT_TEXT
        サブカテゴリ    行番号 (1 - m)
    1.  精度テキスト                    (AwSr::TEXT)
```

データム文字レコード

```
    0.  カテゴリレコード                (AwSr::CATEGORY)
        メインカテゴリ  AwSrCategory::GT_DATUM
        サブカテゴリ    行番号 (1 - m)
    1.  データム文字                    (AwSr::TEXT)
```

幾何公差記入枠線レコード

```
    0.  カテゴリレコード                (AwSr::CATEGORY)
        メインカテゴリ  AwSrCategory::GT_BOX
        サブカテゴリ    行番号 (1 - m)
    1.  サブレコード #3, #4 で構成する線分群
```

引出線レコード

```
    0.  カテゴリ サブレコード           (AwSr::CATEGORY)
        メインカテゴリ  AwSrCategory::LEADER
        サブカテゴリ    #1
    1.  引出線始点                      (AwSr::STARTPOINT)
    2.  引出線の点 (AwSr::LINE)
        :
    m.  引出線の点                      (AwSr::LINE)
    m+. 矢羽マーク                      (AwSr::MARK)
```

m は引出線の点数。32 点以下。

5.11 ハッチングアイテム

アイテムの定義

閉領域内を一定間隔の平行線分でおおう。

サブレコード構造 (バージョン 9 以降)

```
0.      ハッチパラメータ (AwSr::XHTPARAM2)
1.      境界 #1
:
n.      境界 #n
n+1.    End of item(AwSr::EOI)
```

境界はストリングアイテムと同じで AwSr::STARTPOINT,[LINE orCIRCLE or BEZIER]+ の並び。

バージョン 8 以下の古いサブレコード構造 (使用してはいけません。参考のみ)

```
0.      ハッチングラインの始点(AwSr::STARTPOINT)
1.      ハッチングラインの終点(AwSr::LINE)
:
2m-2.   ハッチングラインの始点(AwSr::STARTPOINT)
2m-1.   ハッチングラインの終点(AwSr::LINE)
2m.     End of item(AwSr::EOI)
```

m はハッチングライン数

アイテムの最小／最大座標値

Xmin = MIN (各サブレコードの最小 X)

Ymin = MIN（各サブレコードの最小 Y）
Xmax = MAX（各サブレコードの最大 X）
Ymax = MAX（各サブレコードの最大 Y）

5.12 塗り潰しアイテム

アイテムの定義
閉領域内を指定パターンでおおう。

サブレコード構造
0. 塗り潰しパラメータ(AwSr::AFLPARAM2)
1. 境界 #1
:
n. 境界 #n
n+1. End of Item(AwSr::EOI)

制約
(1) 境界は次のサブレコードの並びである。
AwSr::STARTPOINT、[AwSr::LINE, AwSr::CIRCLE または AwSr::BEZIER]+
(2) 境界は閉じていること（始点と終点が一致すること）。
(3) 境界は複数持つことができる。

アイテムの最小／最大座標値
Xmin = MIN(各サブレコードの最小 X)
Ymin = MIN(各サブレコードの最小 Y)
Xmax = MAX(各サブレコードの最大 X)
Ymax = MAX(各サブレコードの最大 Y)

5.13 メンバアイテム

アイテムの定義
同時設計で使用するサブモデルのこと。本来のサブモデルと区別するために、アイテムタイプ番号が異なる。このアイテムは Advance CAD 実行時のみ存在し、モデルファイルに保存されることはない。

5.14 APG アイテム

アイテムの定義
APG 配置時に作成される図形アイテムの集合である。APG 配置時の APG 名称、APG パラメータ、配置条件などを持っている。APGREGEN で再作成できる。

サブレコード構造
0. ヘッダーブロック
カテゴリレコード (AwSr::CATEGORY)
メインカテゴリ AwSrCategory::APG_HEADER
サブカテゴリ #1
APG 名称 (AwSr::NGTEXT)
APG 配置条件 (AwSr::PLACEMENT)
原点、ボックス、ミラー条件、配置角度

1.APG パラメータブロック
カテゴリレコード (AwSr::CATEGORY)
メインカテゴリ AwSrCategory::APG_HEADER
サブカテゴリ #2

APG パラメータ値 A=105　(AwSr::NGTEXT)
　:

2. 図形データブロック
カテゴリレコード　(AwSr::CATEGORY)
　メインカテゴリ AwSrCategory::APG_BEGIN
　サブカテゴリ　#0

もとのアイテムの属性　(AwSr::ATTRIBUTES2)
もとのアイテムのサブレコード
　:
もとのアイテムの属性　(AwSr::ATTRIBUTES2)
もとのアイテムのサブレコード
カテゴリレコード　(AwSr::CATEGORY)
　メインカテゴリ AwSrCategory::APG_END
　サブカテゴリ　#0

3. End of item(AwSr::EOI)

アイテムの最小／最大座標値
Xmin = MIN(各サブレコードの最小 X)
Ymin = MIN(各サブレコードの最小 Y)
Xmax = MAX(各サブレコードの最大 X)
Ymax = MAX(各サブレコードの最大 Y)

5.15 アソシエイトアイテム

アイテムの定義
　アソシエイトアイテム名と、関係するアイテムのアイテム識別子の集合体からなるアイテム。
　ドローイングレイアウトピクチャには作成しないこと。

サブレコード構造
0. ヘッダーブロック
カテゴリレコード　(AwSr::CATEGORY)
　メインカテゴリ AwSrCategory::ASC_HEADER
　サブカテゴリ　n (アソシエイトカテゴリ番号)
アソシエイトアイテム名　(AwSr::NGTEXT)
カテゴリ　(AwSr::CATEGORY)（オプショナル）
　メインカテゴリ AwSrCategory::ASC_SUBMDL
　サブカテゴリ　#1
シンボル名　(AwSr::NGTEXT)（オプショナル）
カテゴリ　(AwSr::CATEGORY)（オプショナル）
　メインカテゴリ AwSrCategory::ASC_SUBMDL
　サブカテゴリ　#2

X,Y,Z,ANGLE,PIC　(AwSr::SCALAR)（オプショナル）
　X,Y,Z : 原点
　ANGLE : 配置角度 (0,90,180,270)
　PIC　: 配置ピクチャ

1. アソシエイト ブロック
アソシエイト type ,　IDPTR1(AwSr::ASSOCIATION)
アソシエイト type ,　IDPTR2 (AwSr::ASSOCIATION)
:
アソシエイト type ,　IDPTRn(AwSr::ASSOCIATION)

2. End of Item(AwSr::EOI)

オプショナルのレコードはなくてもよい。

5.16 シンボルアイテム

アイテムの定義
属性データ (シンボル名・ベーステキスト・ノードポイント・ノードテキストなど) と任意の図形データの集合体からなるアイテム。

サブレコード構造
0. ヘッダー ブロック
カテゴリ (AwSr::CATEGORY)
メインカテゴリ AwSrCategory::SYM_HEADER
サブカテゴリ #0
シンボル名 (AwSr::NGTEXT)
配置日時 (AwSr::TIMESTAMP)
配置パラメータ (位置、縮尺など)(AwSr::PLACEMENT)

1. ベーステキスト ブロック (オプショナル)
カテゴリ (AwSr::CATEGORY)
メインカテゴリ AwSrCategory::SYM_TEXT_B
サブカテゴリ #0
ベーステキスト (AwSr::TEXT)

2. ノードポイント ブロック (オプショナル)
カテゴリ (AwSr::CATEGORY)
メインカテゴリ AwSrCategory::SYM_NODE
サブカテゴリ ノード数 (n)
ノードポイント #1 (AwSr::POINT)
ノードポイント #2 (AwSr::POINT)
: :
ノードポイント #n (AwSr::POINT)

3. ノードテキスト ブロック (オプショナル)
カテゴリ (AwSr::CATEGORY)
メインカテゴリ AwSrCategory::SYM_TEXT_T または SYM_TEXT_S
サブカテゴリ ノード番号
ノードテキスト (AwSr::TEXT)
: :
カテゴリ (AwSr::CATEGORY)
メインカテゴリ AwSrCategory::SYM_TEXT_T または SYM_TEXT_S
サブカテゴリ ノード番号
ノードテキスト (AwSr::TEXT)

4. 図形データ ブロック
このブロックは、表示図形を構成するサブアイテムを含む。
カテゴリ (AwSr::CATEGORY)
メインカテゴリ AwSrCategory::SYM_BEGIN
サブカテゴリ #0

```
┌ サブアイテムの属性          (AwSr::ATTRIBUTES2)
│ Sub-record              サブアイテムのサブレコード
│    :                        :
│ Sub-record              サブアイテムのサブレコード
└    :                        :
┌ サブアイテムの属性          (AwSr::ATTRIBUTES2)
│ Sub-record              サブアイテムのサブレコード
│    :                        :
│ Sub-record              サブアイテムのサブレコード
└    :                        :
┌ サブアイテムの属性          (AwSr::ATTRIBUTES2)
│ Sub-record              サブアイテムのサブレコード
│    :                        :
└ Sub-record              サブアイテムのサブレコード

  カテゴリ                (AwSr::CATEGORY)
     メインカテゴリ  AwSrCategory::SYM_END
     サブカテゴリ   #0
5. End of item  (AwSr::EOI)
```

制約

(1) ベーステキスト ブロック、ノードポイント ブロックまたはノードテキストブロックのないシンボルもある。

(2) 通常表示されるのは、図形データブロックだけである。ただし、アイデントされたときは、ノードポイントも表示される。

(3) シンボルはネスティングできない。

アイテムの最小／最大座標値

Xmin = MIN(各サブレコードの最小 X)
Ymin = MIN(各サブレコードの最小 Y)
Xmax = MAX(各サブレコードの最大 X)
Ymax = MAX(各サブレコードの最大 Y)

5.17 サブモデルアイテム

アイテムの定義

既存モデルの１つのピクチャ上のアイテムで構成されたアイテム。

サブレコード構造

```
0.ヘッダー ブロック
   カテゴリ                    (AwSr::CATEGORY)
      メインカテゴリ  AwSrCategory::SUB_HEADER
      サブカテゴリ   Bit flags
   元のピクチャ番号          (AwSr::SCALAR)
   サブモデル名              (AwSr::NGTEXT)
   配置日時                  (AwSr::TIMESTAMP)
   配置パラメータ（位置、縮尺など)(AwSr::PLACEMENT)

   以下のサブモデル配置時マスクはオプショナル。
   カテゴリ                    (AwSr::CATEGORY)
      メインカテゴリ  AwSrCategory::SUB_MASK
      サブカテゴリ   #1( アイテムタイプマスク )
   サブモデルのマスクデータ(AwSr::SCALAR)
   カテゴリ                    (AwSr::CATEGORY)
      メインカテゴリ  AwSrCategory::SUB_MASK
      サブカテゴリ   #2( クラスマスク )
```

サブモデルのマスクデータ(AwSr::SCALAR)
カテゴリ (AwSr::CATEGORY)
メインカテゴリ AwSrCategory::SUB_MASK
サブカテゴリ #3(レビジョンマスク)
サブモデルのマスクデータ(AwSr::SCALAR)
カテゴリ (AwSr::CATEGORY)
メインカテゴリ AwSrCategory::SUB_MASK
サブカテゴリ #4(線種マスク)
サブモデルのマスクデータ(AwSr::SCALAR)
カテゴリ (AwSr::CATEGORY)
メインカテゴリ AwSrCategory::SUB_MASK
サブカテゴリ #5(線幅マスク)
サブモデルのマスクデータ(AwSr::SCALAR)

1. アイテムデータ ブロック
このブロックは、サブモデルを構成するサブアイテムを含む。
カテゴリ (AwSr::CATEGORY)
メインカテゴリ AwSrCategory::SUB_BEGIN
サブカテゴリ #0

```
Item attribute          サブアイテムの属性      (AwSr::ATTRIBUTES2)
Sub-record              サブアイテムのサブレコード
  :                       :
Sub-record              サブアイテムのサブレコード
```

```
Item attribute          サブアイテムの属性      (AwSr::ATTRIBUTES2)
Sub-record              サブアイテムのサブレコード
  :                       :
Sub-record              サブアイテムのサブレコード
  :                       :
Item attribute          サブアイテムの属性      (AwSr::ATTRIBUTES2)
Sub-record              サブアイテムのサブレコード
  :                       :
Sub-record              サブアイテムのサブレコード
```

カテゴリ (AwSr::CATEGORY)
メインカテゴリ AwSrCategory::SUB_END
サブカテゴリ #0

2.End of item (AwSr::EOI)

アイテムの最小／最大座標値
Xmin = MIN(各サブレコードの最小 X)
Ymin = MIN(各サブレコードの最小 Y)
Xmax = MAX(各サブレコードの最大 X)
Ymax = MAX(各サブレコードの最大 Y)

5.18 イメージアイテム

アイテムの定義
イメージを表現するアイテム。

サブレコード構造
0. Image(AwSr::IMAGE)
1. End of item(AwSr::EOI)

アイテムの最小／最大座標値

Xmin = Image を含む最小矩形の最小 X
Ymin = Image を含む最小矩形の最小 Y
Xmax = Image を含む最小矩形の最大 X
Ymax = Image を含む最小矩形の最大 Y

制約

このアイテムの先頭は Image サブレコードをおきます。

このアイテムは、ピクチャの背景として配置したイメージとして使用します。背景であるため、通常のコマンドではピックできません。イメージアイテムの配置、修正、削除のコマンドがあります。

ユーザプログラミングでは、このアイテムを修正しないで下さい。

第 6 章　サブレコード

サブレコードはアイテムを構成する最小単位です。下記の表はサブレコードタイプの一覧です。Version18 では、SXF の仕様を取り込み、サブレコードタイプの見直しを行ないました。その結果、いくつかのサブレコードタイプが廃止となり、新しいサブレコードタイプが追加されました。表中の区分に「廃止」と記したサブレコードタイプは使用できません。Version 17 以下で作成されたモデルファイルを読み込むと、自動的に適切な新しいサブレコードに変換します。
区分に「NC」と記したサブレコードは NC のみで使用します。これはアイテムに付加するものではなく、モデルファイルには保存できません。
サブレコードタイプは整数番号ですが、それを表すサブレコードタイプ定数も示します。タイプ定数はヘッダーファイル AwSr.h で定義しています。本書中の関数でサブレコードタイプを指示するものがあります。その場合はサブレコード番号を記入するよりはタイプ定数を記入する方が分かりやすいでしょう。

サブレコード一覧表

番号	サブレコード用途	タイプ定数	区分
1	カテゴリ	AwSr::CATEGORY	汎用
2	点	AwSr::POINT	幾何
3	始点	AwSr::STARTPOINT	幾何
4	線分	AwSr::LINE	幾何
5	円、円弧	AwSr::CIRCLE	幾何
6	Bezier 曲線	AwSr::BEZIER	幾何
7 － 10	（未使用）		
11	テキストパラメータ (-V17)		廃止
12	テキスト／マーク 原点枠 (-V17)		廃止
13	文字列 (-V17)		廃止
14	マークパラメータ (-V17)		廃止
15	寸法アイテム種類 (-V17)		廃止
16	寸法アイテム基準データ (-V17)		廃止
17	幾何公差枠	AwSr::GTFRAME	製図
18	塗り潰しパラメータ (-V17)		廃止
19	ハッチングパラメータ (-V17)		廃止
20	プロパティ（特性データ）	AwSr::PROPERTY	汎用

番号	サブレコード用途	タイプ定数	区分
21	NC マシニングレコード		NC
22	3D Position (x,y,z)		NC
23	(system use -V18)		廃止
24	(system use -V18)		廃止
25	スカラ列	AwSr::SCALAR	汎用
26	非図形文字列	AwSr::NGTEXT	汎用
27	日付	AwSr::TIMESTAMP	構造化
28	構造化アイテム配置パラメータ	AwSr::PLACEMENT	構造化
29	アソシエイション	AwSr::ASSOCIATION	構造化
30	元のアイテム属性 (-V17)		廃止
31	アイテムの終端	AwSr::EOI	汎用
32	元のアイテム属性 (V18-)	AwSr::ATTRIBUTES2	構造化
33	文字列 (V18-)	AwSr::TEXT	製図
34	マーク (V18-)	AwSr::MARK	製図
35	塗り潰しパラメータ (V18-)	AwSr::AFLPARAM2	製図
36	ハッチングパラメータ (V18-)	AwSr::XHTPARAM2	製図
37	寸法パラメータ (V18-)	AwSr::DIMENSION	製図
38	イメージ (V19-)	AwSr::IMAGE	SXF
39	作図部品定義 (V18-)	AwSr::SXFSFIGORG	SXF
40	作図部品配置 (V18-)	AwSr::SXFSFIGLOC	SXF
41	作図部品配置終了 (V18-)	AwSr::SXFSFIGEND	SXF
42	AFL/XHT 境界表示制御 (V18-)	AwSr::SXFBNDATTR	SXF
43	元図定義 (V18-)	AwSr::SXFORGDEF	SXF
44	元図参照 (V18-)	AwSr::SXFORGREF	SXF
45	3D (V19-)	AwSr::GEOM3D	3D

6.1 抽象クラス AwSr

サブレコードは C++ クラスで実装しています。全てのサブレコードが抽象クラス AwSr を継承します。AwSr クラスは全てのサブレコードの具象クラスに共通するメソッドを持ちます。

6.1.1 サブレコードタイプ

次のメソッドはサブレコードのタイプを得るメソッドです。

```
int GetId() const;
```

戻り値　このオブジェクトのサブレコードタイプを返します。

サブレコードタイプから具象クラスがわかります。サブレコードタイプごとに具象クラスがあります。
次のコードフラグメントはサブレコードの抽象クラスを具象クラスにダウンキャストする例です。

```
for (int i = 0; i < pItem->GetSrCount(); ++i) {
  const AwSr* pSr = pItem->GetSrAt(i);
  if (pSr->GetId() == AwSr::STARTPOINT) {
    const AwSrStart* pSrStart = static_cast<const AwSrStart*>(pSr);
    // AwSrStart クラスのメソッドを使うことができる。
  } else if (pSr->GetId() == AwSr::LINE) {
    const AwSrLine* pSrLine = static_cast<const AwSrLine*>(pSr);
    // AwSrLine クラスのメソッドを使うことができる。
  }
}
```

サブレコードタイプが AwSr::STARTPOINT であれば、これは曲線の始点サブレコードで、それの具象クラスは AwSrStart クラスです。次の文は static_cast を使って const AwSr* を const AwSrStart* にダウンキャストします。

```
const AwSrStart* pSrStart = static_cast<const AwSrStart*>(pSr);
```

このようにして得たポインタ pSrStart を使うと AwSrStart クラスのメソッドを使うことができます。
このとき、間違った具象クラスにダウンキャストしないようにプログラミングしなければなりません。
dynamic_cast を使うと間違った具象クラスにダウンキャストしようとすると null ポインタを返します。
dynamic_cast は実行時にクラスの継承を調べるので正確ですが、実行時の負荷が大きくなります。

GetId() メソッドを使えばできることですが、簡便のために用意されたメソッドがあります。
次のメソッドはサブレコードが曲線サブレコードタイプ（AwSr::LINE、AwSr::CIRCLE または AwSr::BEZIER）か判定します。

```
bool IsCurve() const;
```

戻り値　曲線サブレコードタイプなら true を返します。

6.1.2　サブレコードの線種

通常はアイテムの線種で表示しますが、それとは異なる線種で表示したいときに使用します。サブレコードの線種番号は次のようになります。

0	アイテムと同じ。
1	非表示。
2 － 64	アイテムの線種番（1 - AwItemAttributes::MAX_LINEFONT）に対応する。

次のメソッドはサブレコードの線種を取得／設定するメソッドです。

```
int  GetLineFont(bool bExternal = false) const;
```

bExternal　線種番号変換スイッチ。
戻り値　線種番号を返します。

```
void SetLineFont(int font, bool bExternal = false);
```

font　線種番号。
bExternal　線種番号変換スイッチ。

引数 bExternal を true とすると下記のように変換した線種番号になります。

-1	非表示。
0	アイテムと同じ。
1 － 63	アイテムの線種番号（1 - AwItemAttributes::MAX_LINEFONT）。

簡便のために用意されたメソッドがあります（Get/Set メソッドでもできる）。
このサブレコードは表示か判定します。

```
bool IsVisible() const;
```

戻り値　非表示でないとき true を返します。

このサブレコード線種を「アイテムと同じ」に設定します。

```
void InheritLineFont();
```

このサブレコードを非表示にします。
```
void MakeInvisible();
```

6.1.3 サブレコードの線幅

通常はアイテムの線幅で表示しますが、それとは異なる線幅で表示したいときに使用します。サブレコードの線幅番号は次のようになります。

0	アイテムと同じ。
1 － 16	アイテムの線幅番号 (1- AwItemAttributes::MAX_LINEWEIGHT) に対応する。

次のメソッドはサブレコードの線幅を取得／設定するメソッドです。
```
int  GetLineWeight() const;
void SetLineWeight(int weight);
```

簡便のために用意されたメソッドがあります（Set メソッドでもできる)。
このサブレコード線幅を「アイテムと同じ」に設定します。
```
void InheritLineWeight();
```

6.1.4 サブレコードのデータ

サブレコードのデータは具象クラスごとに異なります。具象クラスにはデータの取得／設定メソッド (アクセッサと呼ぶ) があり、それを使いますので、サブレコードのデータタイプ／データ数を使うことはほとんどありません。

データタイプは下記のどれかです。

AwSr::CHAR	char (8 bits)。
AwSr::SHORT	short (16 bits)。
AwSr::INT	int (32 bits)。
AwSr::FLOAT	float (32 bits)。
AwSr::DOUBLE	double (64 bits)。
AwSr::MIXED	**上記データタイプの複合。**

データタイプを得るメソッドです。
```
int GetDataType() const;
```
戻り値　データタイプを返します。

データ数を得るメソッドです。
```
int GetDataCount() const;
```
戻り値　データ数を返します。

次のメソッドはサブレコードのデータの値が正しいか判定します。
```
bool IsValid();
```
戻り値　サブレコードが有効と判断すると true を返します。

6.1.5 オブジェクトのコピイ

AwSr クラスは抽象クラスなので、オブジェクトのコピイを作るには、具象クラスを調べてコピイしなければなりません。これを間単に行えるように、サブレコードのオブジェクトをコピイするメソッドがあります。
```
AwSr* Clone() const;
```
戻り値　具象クラスのオブジェクトのコピイを作りそのポインタを返します。このメソッドで得たオブジェクトは不要になった時点で delete しなければなりません。

ここで注意をひとつ。AwSr クラスの代入演算子 (=) は使用できません。AwSr* のポインタ p Sr1、pSr2 に対して次のような代入を行うとコンパイルエラーになるようにしています。

```
*pSr2 = *pSr1; // オブジェクトの内容をコピイしたいができない。
```

この代入文は、AwSr クラスのデータだけをコピイし具象クラスで定義したデータはコピイしないのに、あたかも具象クラスのデータもコピイすると勘違いしやすいからです。
ポインタ変数の代入はできます。次の文は pSr2 が pSr1 と同じオブジェクトを指すようにします。

```
pSr2 = pSr1; // ポインタの代入。
```

もし pSr1 と pSr2 の指すオブジェクトの具象クラスが同じなら、次のようにダウンキャストすれば代入演算子が使えます（オブジェクトの内容をコピイ）。

```
static_cast<AwSrStart&>(*pSr2) = static_cast<AwSrStart&>(*pSr1); // 両方 AwSrStart オブジェクト
```

6.1.6 座標変換

座標を持ち座標変換が定義されるサブレコードの具象クラスは下記のメソッドをオーバライドします。それ以外の具象クラスでは何もしないメソッドです。

このサブレコードの座標を平行移動します。

```
void Translate(const G2Point& vector);
```

vector　移動量を与えます。サブレコードの座標 += 移動量となります。

このサブレコードの座標を変換します。

```
void Transform(const G2Matrix& matrix);
```

matrix　座標変換行列オブジェクト。座標変換は単一スケーリング、回転、平行移動を設定することができます。反転は設定できません。

6.2 抽象クラス AwSrCurve

このクラスは抽象クラス AwSr を継承した抽象クラスです。曲線サブレコードに共通なメソッドのインターフェイスを定義します。AwSrLine、AwSrCircle と AwSrBezier クラスは AwSrCurve クラスを継承し下記のメソッドを実装します。

この曲線サブレコードの終点を得ます。

```
G2Point GetEndPoint() const;
```

この曲線サブレコードに対応する曲線オブジェクトを得ます。

```
G2Curve* GetCurve(const G2Point& start) const;
```

start　この曲線サブレコードの始点を与えます。
戻り値　曲線オブジェクトへのポインタを返します。このメソッドで得たオブジェクトは不要になった時点で削除 (delete) しなければなりません。

6.3 カテゴリ AwSrCategory

このサブレコードは、これに続く一群のサブレコード（レコードと呼ぶ）の役割を表します。ひとつのレコードは、カテゴリサブレコード で始まり、次のカテゴリ サブレコードまたはアイテムの終了サブレコード (AwSr::EOI) の直前までです。

- (省略)
- サブレコード (AwSr::CATEGORY)
- サブレコードの並び
- サブレコード (AwSr::CATEGORY)
- サブレコードの並び
- (省略)

このサブレコードはメジャーコードとマイナーコードの 2 つの整数を持ちます。メジャーコードは 1-255 の値で 0 は使用しません。マイナーコードは 0-255 の値でメジャーコードに依存します。

以下にメジャーコードとマイナーコードを列挙します。メジャーコードを表す定数はヘッダーファイル AwSrCategory.h で定義しています。

```
DIM_ARROW : 寸法線 (Dimension)
      1 = 第１寸法線
      2 = 第２寸法線
      3 = 第３寸法線
      4 = 第４寸法線
LEADER : 引出線 (Drafting)
      切断線のライン (Cutting plane line)
TEXT : グラフィックステキスト (Drafting)
MARK : マーク (General-mark)
DIM_WITNESS :寸法補助線 (Dimension)
      1 = 第１寸法補助線
      2 = 第２寸法補助線
BALLOON ; 風船 (Reference label/note)
      0 = 風船用テキスト
      1 = 風船用マーク
RAD_LEADER : 半径寸法の円中心側引出線 (DMR dimension)
GT : 幾何公差 (Geometrical tolerance)
      許容値の行数
WELD : 溶接記号 (Weld mark)
      1 = 矢の手前側溶接記号のマーク
      2 = 矢の向う側溶接記号のマーク
      3 = 全周／現場溶接記号のマーク
      4 = 溶接部の表面形状マーク（手前側）
      5 = 溶接部の表面形状マーク（向う側）
      11 = 溶接部断面寸法または強さ（手前側）
      12 = ルート開先，開先角度（手前側）
      13 = 溶接長さ，数，ピッチ（手前側）
      14 = 溶接部の仕上方法（手前側）
      21 = 溶接部断面寸法または強さ（向う側）
      22 = ルート開先，開先角度（向う側）
      23 = 溶接長さ，数，ピッチ（向う側）
      24 = 溶接部の仕上方法（向う側）
      31 = 特記事項
TEXTURE : 面の肌記号 (Surface texture)
      1 = 面の指示記号のマーク
      11 = 中心線平均粗さの値 (Ra)
      12 = カットオフ値
      13 = 最大長さ (Rmax) ＋点平均粗さ (Rz)
      14 = 基準長さ
      15 = 加工方法
      16 = 筋目方向
      17 = 仕上げ代 / 削り代
      18 = 表面うねり仕様
      19 = 表面性状パラメータ
      20 = ２番目のパラメータ
      21 = ３番目のパラメータ
DIM_TEXT : 寸法値 (Dimension)
      0 = 寸法値テキスト
      1 = 寸法付加テキスト
TOLERANCE : 寸法許容差 (Dimension)
      1 = ± 寸法許容差
      2 = 上の寸法許容差
      3 = 下の寸法許容差
GT_MARK : 幾何公差の記号 (Geometrical tolerance)
GT_TEXT : 幾何公差の精度 (Geometrical tolerance)
```

GT_DATUM : 幾何公差のデータム (Geometrical tolerance)
GT_BOX : 幾何公差記入枠 (Geometrical tolerance)
CPL_TEXT : 切断線のテキスト (Cutting plane line)
　　1 = 始点側
　　2 = 終点側
CPL_MARK : 切断線のマーク (Cutting plane line)
　　1 = 始点側
　　2 = 終点側
CPL_LINE : 切断線 (Cutting plane line)
TEXTBOX : テキスト枠の線 (Drafting)
UNDERLINE : テキスト下線 (Drafting)
　　n = テキスト下線の数
NC_DRILL : NC Drill
NC_EDM : NC EDM
PROP_ID : 特性データカテゴリ番号
　　n = カテゴリ番号
PROP_TAG : 特性データ TAG タイプ
　　1 = GNT
　　2 = 風船
　　3 = TAG
ASC_HEADER : アソシエイトアイテムの名前とアソシエイト分類番号
　　n = アソシエイト分類番号
ASC_SUBMDL : アソシエイト属性
　　1 = サブモデル名
　　2 = 配置原点
ASC_MOVE : CUT_MOVE, CUT_REGEN 用オリジナルデータ。
　　直後の サブレコード (AwSr::SCALAR) に元の点座標 (x,y)、移動先点座標 (x,y), スケール、出力先ピクチャ番号および境界上のアイテムの処理方法を持つ (7 つの実数)。
AREA : Measure 面積データ
SYM_HEADER : シンボルヘッダ (Symbol)
SYM_TEXT_B : シンボルベーステキスト (Symbol)
SYM_NODE　: シンボルノード点 (Symbol)
SYM_TEXT_N : シンボル ノードテキスト #1 (ノード Name テキスト)
SYM_TEXT_T : シンボル ノードテキスト #2 (コネクトノード プレイスメント)
SYM_TEXT_S : シンボル ノードテキスト #3 (コネクトノード シーケンス番号)
SYM_BEGIN　: シンボルデータの開始 (Symbol)
SYM_END　　: シンボルデータの終了 (Symbol)
SUB_HEADER : サブモデルヘッダ (Submodel)
　　bit　9 : 寸法要素再作成フラグ
　　bit 10 : 製図要素縮尺フラグ
　　bit 11 : ピクチャ参照フラグ
SUB_MASK : サブモデルのマスクデータ (Submodel)
　　1 = アイテムマスク
　　2 = クラスマスク
　　3 = レビジョンマスク
　　4 = 線種マスク
　　5 = 線幅マスク
SUB_BEGIN : サブモデルデータの開始 (Submodel)
SUB_END　: サブモデルデータの終了 (Submodel)
APG_HEADER : APG ヘッダ
　　1 = APG ファイル名および配置パラメータ
　　2 = APG 変数の値
APG_BEGIN : APG データの開始。
APG_END　: APG データの終了。

AwSrCategory クラスはカテゴリサブレコードを表現するクラスです。このサブレコードは表示しませんので、サブレコード線種、線幅は意味を持ちません。

6.3.1 コンストラクタ

```
AwSrCategory();
```
メジャーコードとマイナーコードは 0 です。
```
AwSrCategory(int major, int minor);
```
メジャーコード (major) とマイナーコード (minor) コードを与える。

6.3.2 アクセッサ

メジャーコードとマイナーコードの組み合わせの検査は行いません。
メジャーコードを取得／設定します。
```
int  GetMajorCode() const;
void SetMajorCode(int value);
```

マイナーコードを取得／設定します。
```
int  GetMinorCode() const;
void SetMinorCode(int value);
```

6.4 点 AwSrPoint

AwSrPoint クラス は点サブレコードを表現するクラスです。
このサブレコードは点の座標 (x,y) を倍精度実数で持ちます。
アイテムの線種は点を表示する形状を表します。
- アイテム線種　1: プラス（＋）
- アイテム線種　2: ダイアモンド（◇）
- アイテム線種　3: ドット

サブレコードの線種は点を表示する形状と解釈します。
- サブレコード線種　0:アイテムの線種に従う。
- サブレコード線種　1:アスタリスク（*）
- サブレコード線種　2:プラス（＋）
- サブレコード線種　3:ダイアモンド（◇）
- サブレコード線種　4:ドット

線種、線幅は常に 1 で表示します。

6.4.1 コンストラクタ

```
AwSrPoint();
```
座標 (0, 0) のオブジェクト。
```
AwSrPoint(const G2Point& point, int font = 0, int weight = 0);
```
点、線種、線幅を指定するコンストラクタ。

6.4.2 アクセッサ

点を取得／設定します。
```
G2Point GetPoint() const;
void SetPoint(const G2Point& point);
void SetPoint(double x, double y);
```

6.5 始点 AwSrStart

AwSrStart クラス は始点サブレコードを表現するクラスです。

このサブレコードは曲線の始点座標 (x,y) を倍精度実数で持ちます。このサブレコードの次には必ず曲線サブレコード（線分、円弧、Bezier 曲線）が続かなければなりません。
このサブレコードは表示しませんので、サブレコード線種、線幅は意味を持ちません。

6.5.1 コンストラクタ

AwSrStart();
　座標 (0, 0) のオブジェクト。
AwSrStart(const G2Point& ps);
　点を指定するコンストラクタ。

6.5.2 アクセッサ

始点を取得／設定します。
G2Point GetStartPoint() const;
void SetStartPoint(const G2Point& point);

6.6 線分 AwSrLine

AwSrLine クラス は線分サブレコードを表現するクラスです。
このサブレコードは線分の終点座標 (x,y) を倍精度実数で持ちます。この線分の始点は直前の始点サブレコードか、直前の曲線サブレコード（線分、円弧、Bezier 曲線）の終点です。

6.6.1 コンストラクタ

AwSrLine();
AwSrLine(const G2Point& pe, int font = 0, int weight = 0);
AwSrLine(const G2Line& line, int font = 0, int weight = 0);

6.6.2 アクセッサ

終点を取得／設定します。
G2Point GetEndPoint() const;
void SetEndPoint(const G2Point& end);

線分 (G2Line) を取得／設定します。
G2Line Get(const G2Point& start) const;
void Set(const G2Line& line);

6.7 円／円弧 AwSrCircle

AwSrCircle クラス は円弧サブレコードを表現するクラスです。
このサブレコードは円弧データを倍精度実数で持ちます。この円弧の始点は直前の始点サブレコードか、直前の曲線サブレコード（線分、円弧、Bezier 曲線）の終点です。

- 中点座標 (xm,ym)。中間点は必ず弧の２分点になければならない。（将来除去する予定）
- 終点座標 (xe,ye)。円は始点と同じ座標。
- 中心点座標 (xc,yc)。
- 半径。
- 中心角 (Radian, -2π から 2π)。
  正の値は 反時計廻り (G2Math::CCw)、負の値は 時計廻り (G2Math::Cw) を表す。

6.7.1 コンストラクタ

```
AwSrCircle();
AwSrCircle(const G2Circle& carc, int font = 0, int weight = 0);
```

6.7.2 アクセッサ

円弧のデータを取得するメソッドです。

```
double GetRadius() const;
```
　戻り値　半径を返します。

```
double GetAngleExtent() const;
```
　戻り値　円弧の中心角を返します。

```
double GetLength() const;
```
　戻り値　円弧長を返します。

```
G2Point GetCenter() const;
```
　戻り値　円の中心点返します。

```
G2Point GetEndPoint() const;
```
　戻り値　円弧の終点を返します。

円弧 (G2Circle) を取得／設定します。

```
G2Circle Get(const G2Point& start) const;
void Set(const G2Circle& carc);
```

6.8 Bezier 曲線 AwSrBezier

AwSrBezier クラスは Bezier 曲線サブレコードを表現するクラスです。
このサブレコードは 3 次 Bezier 曲線データを倍精度実数で持ちます。この曲線の始点は直前の始点サブレコードか、直前の曲線サブレコード（線分、円弧、Bezier 曲線）の終点です。

- **制御点 1 の座標 (x1,y1)**
- **制御点 2 の座標 (x2,y2)**
- **終点座標 (x3,y3)**
- **曲線長さ**

制御点列 Q0, Q1, Q2, Q3 として、曲線の点 P(t) は次の式で表します。

```
P(t) = (1-t)^3*Q0 + 3t(1-t)^2*Q1 + 3t^2(1-t)*Q2 + t^3*Q3
(0 <= t <= 1)
```

6.8.1 コンストラクタ

```
AwSrBezier();
AwSrBezier(const G2Bezier& bzc, int font = 0, int weight = 0);
```

6.8.2 アクセッサ

Bezier 曲線データを取得するメソッドです。

```
double GetLength() const;
```
　戻り値　曲線長を返します。

```
G2Point GetEndPoint() const;
```
　戻り値　曲線の終点を返します。

Bezier 曲線 (G2Bezier) を取得／設定します。

```
G2Bezier Get(const G2Point& start) const;
void Set(const G2Bezier& bzc);
```

6.9 プロパティ AwSrProperty

AwSrProperty クラス はプロパティサブレコードを表現するクラスです。

このサブレコードはヘッダとトレイラのふたつを組で使います。このヘッダとトレイラの間に特性データを挟みます。組となるヘッダとトレイラは同じカテゴリコードでなければなりません。プロパティのカテゴリコードは 1-255 の値で 0 は使用しません。

- (省略)
- プロパティヘッダ (AwSr::PROPERTY)
- 特性データを表すサブレコード の並び
- プロパティトレイラ (AwSr::PROPERTY)
- (省略)

このサブレコードは表示しませんので、サブレコード線種、線幅は意味を持ちません。

6.9.1 コンストラクタ

```
AwSrProperty();
AwSrProperty(int categ, int type);
```

6.9.2 アクセッサ

プロパティヘッダ／トレイラを判定するには次のメソッドを使います。

```
bool IsHeader() const;
```

戻り値　このサブレコードがヘッダであるとき true、トレイラのとき false を返します。

このサブレコードをトレイラにするには次のメソッドを使います。

```
void SetTrailer(bool b = true);
```

戻り値　このサブレコードをトレイラにするとき引数 true を渡します。

カテゴリコードを取得／設定します。

```
int  GetCategory() const;
void SetCategory(int categ);
```

特性 データのレコード番号 (1 - 255)

6.10 スカラ配列 AwSrScalar

AwSrScalar クラスはスカラ列サブレコードを表現するクラスです。

このサブレコードは数値列を保持するのに使います。数値の種類と数値列の長さの上限を下記に示します。ひとつのサブレコードに種類が異なる数値を混在させることはできません。

AwSr::DUBLE	double (64 bits)	1 - AwSr::MAX_DOUBLE_LENGTH
AwSr::FLOAT	float (32 bits)	1 - AwSr::MAX_FLOAT_LENGTH
AwSr::INT	int (32 bits)	1 - AwSr::MAX_INT_LENGTH
AwSr::SHORT	short (16 bits)	1 - AwSr::MAX_SHORT_LENGTH

このサブレコードは表示しませんので、サブレコード線種、線幅は意味を持ちません。

6.10.1 コンストラクタ

数値のデータタイプごとのコンストラクタがあります。オブジェクト生成後はデータタイプ、データ数は変更できません。

```
AwSrScalar(const double darr[], int count);
```

```
AwSrScalar(const float farr[], int count);
AwSrScalar(const int iarr[], int count);
AwSrScalar(const short sarr[], int count);
```

6.10.2 アクセッサ

次のメソッドは数値のデータタイプとデータ数を得るメソッドです。

```
int GetDataType() const;
int GetDataCount() const;
```

次のメソッドは指定した位置の数値を取得します。

```
double GetDoubleAt(int index) const;
float  GetFloatAt(int index) const;
int    GetIntAt(int index) const;
short  GetShortAt(int index) const;
```

インデクスが範囲外のときは 0 を返します。

次のメソッドは指定した位置に数値を設定します。

```
void SetDoubleAt(int index, double value);
void SetFloatAt(int index, float value);
void SetIntAt(int index, int value);
void SetShortAt(int index, short value)
```

6.10.3 まとめてコピイ

次のメソッドは配列に数値をコピイします。

```
int GetDoubles(int from, int count, double darr[]) const;
int GetFloats(int from, int count, float farr[]) const;
int GetInts(int from, int count, int iarr[]) const;
int GetShorts(int from, int count, short sarr[]) const;
```

from	最初の数値のインデクス (0 <= from < GetDataCount())。
count	コピイする数値の数 (1 <= count)。
戻り値	引数の配列にコピイした数値の数。

6.11 非図形文字列 AwSrNgText

AwSrNgText クラスは非図形文字列サブレコードを表現するクラスです。このサブレコードは表示しませんので、サブレコード線種、線幅は意味を持ちません。

このサブレコードは文字列を保持するのに使います。

6.11.1 コンストラクタ

```
AwSrNgText();
```

空文字列です。

```
AwSrNgText(const std::string& text);
```

文字列 (text) を渡すコンストラクタ。

6.11.2 アクセッサ

文字コードは日本語 EUC です。文字列長さは AwSr::MAX_BYTE_LENGTH（バイト）以下です。長い文字列を与えると上限を超える部分は切断します。切断で日本語文字（1 文字複数バイト）の片割れバイトができないよう注意してください。

文字列を取得／設定します。
```
const std::string& GetText() const;
void SetText(const std::string& text);
```

空文字列のオブジェクトは無効です。つぎのメソッドは空文字列だと false を返します。
```
bool IsValidData() const;
```

6.11.3 文字列比較

```
int  Compare(const std::string& text) const;
int  Compare(const AwSrNgText& sr) const;
```
戻り値　比較結果 (std::string.compare() と同じ)。

負	このオブジェクトの文字列 < 引数の文字列。
0	ふたつの文字列が等しい。
正	このオブジェクトの文字列 > 引数の文字列。

6.12 文字列 AwSrText

AwSrText クラス はグラフィックステキストサブレコードを表現するクラスです。

このサブレコードはグラフィックステキストのパラメータ、位置、文字列を持ちます。ジェネラルノートの文字列、リファレンスラベルの風船内の文字列や寸法アイテムの寸法値などに使用します。

サブレコード線種は表示 / 非表示の制御のみに使用し、線種は常に実線で表示します。線幅は有効です。

6.12.1 コンストラクタ

```
AwSrText();
```
空文字列。パラメータはなにも設定しません。

6.12.2 パラメーター括設定

AwDrafParam オブジェクトのテキストパラメータをこのオブジェクトに一括して設定するメソッドがあります。
通常の文字列用に設定します。
```
void SetTextParams(const AwDrafParam& param);
```

寸法アイテム用に設定します。
```
void SetForDimension(const AwDrafParam& param);
```

寸法アイテムの寸法許容差用に設定します。
```
void SetForTolerance(const AwDrafParam& param, bool bilateral = false);
```
bilateral　± の寸法許容差用は true、上または下の寸法許容差用は false です。

6.12.3 アクセッサ

ASCII テキストフォント番号を取得／設定します。
```
int  GetAsciiFontId() const;
void SetAsciiFontId(int id);
```
1-99 = 英数字ストロークフォント。
102-109 = アウトラインフォントの英数字部分を使う。

111-130 = トゥルータイプフォントの英数字部分を使う (OS 依存、OS 間の互換性なし)。

日本語テキストフォント番号を取得／設定します。
```
int  GetJapaneseFontId() const;
void SetJapaneseFontId(int id);
```
101 = 日本語ストロークフォント。
102-109 = アウトラインフォント。
111-130 = トゥルータイプフォント (OS 依存、OS 間の互換性なし)。

文字高さを取得／設定します。
```
double GetCharHeight() const;
void   SetCharHeight(double height);
```
ドローイングレイアウトスペースでの文字高さ (0.01 - 327.67)。

文字列角度を取得／設定します。
```
double GetTextAngle() const;
void   SetTextAngle(double angle)
```
角度 (0 － 360.0 度)。

文字列表示モードを取得／設定します。
```
int  GetRotateInternalText() const;
void SetRotateInternalText(int flag);
```
0= 横書き、1= 縦書き。

X 座標ミラーフラッグを取得／設定します。
```
int  GetTextMirrorX() const;
void SetTextMirrorX(int flag);
```
0= しない、1= テキストの Y 軸に対して反転する。

Y 座標ミラーフラッグを取得／設定します。
```
int  GetTextMirrorY() const;
void SetTextMirrorY(int flag);
```
0= しない、1= テキストの X 軸に対して反転する。

文字傾き角を取得／設定します。
```
double GetCharSlantAngle() const;
void   SetCharSlantAngle(double angle);
```
角度 (-85.00 ～ 85.00、0.01 刻み)。

文字縦横比を取得／設定します。
```
double GetCharAspectRatio() const;
void   SetCharAspectRatio(double ratio);
```
文字高さを基準とした比率 (= 幅 / 高さ) (0.1 - 10.0)。

文字列表示水平基準を取得／設定します。
```
int  GetTextJustification() const;
void SetTextJustification(int code);
```
BASE_LEFT= 左詰め、BASE_CENTER= 中央、BASE_RIGHT= 右詰め。

行幅整列係数を取得／設定します。
```
int  GetWidthFill() const;
void SetWidthFill(int code);
```
0 ～ 50% (1% 刻み), 55% ～ 100% (5% 刻み)。

水平方向原点基準を取得／設定します。
```
int  GetTextHorizontalOrigin() const;
void SetTextHorizontalOrigin(int code);
```

BASE_LEFT= 左、BASE_CENTER= 中央、BASE_RIGHT= 右。

垂直方向原点基準を取得／設定します。
```
int  GetTextVerticalOrigin() const;
void SetTextVerticalOrigin(int code);
```
BASE_BOTTOM= 下、BASE_CENTER= 中央、BASE_TOP= 上。

水平方向ゆとり幅を取得／設定します。
```
double GetTextboxHorizontalMargin() const;
void   SetTextboxHorizontalMargin(double margin);
```
ドローイングレイアウトスペースでのゆとり幅 (0.0 ～ 327.67、0.01 刻み)。

垂直方向ゆとり幅を取得／設定します。
```
double GetTextboxVerticalMargin() const;
void   SetTextboxVerticalMargin(double margin);
```
ドローイングレイアウトスペースでのゆとり幅 (0.0 ～ 327.67、0.01 刻み)。

文字間隔を取得／設定します。
```
double GetInterCharSpaceRatio() const;
void   SetInterCharSpaceRatio(double ratio);
```
文字高さを基準とした比率 (= 文字間隔 / 高さ）(0.0 - 10.0)。

行間隔を取得／設定します。
```
double GetInterLineSpaceRatio() const;
void   SetInterLineSpaceRatio(double ratio);
```
文字高さを基準とした比率 (= 行間隔 / 高さ）(0.0 - 10.0)。

文字列枠／下線表示スタイルを取得／設定します。
```
int  GetTextboxStyle() const;
void SetTextboxStyle(int code);
```
STYLE_NONE= なし、STYLE_BOX= 枠、STYLE_UNDERLINE= 下線、
STYLE_BOX_UNDERLINE= 枠と下線、STYLE_VARIABLE_LINE= 可変長下線、
STYLE_DOUBLE_LINE = 二重下線。

6.12.4 文字列

文字コードは日本語 EUC です。使用できる文字に関する詳細は AwEucEncoder クラスの説明を参照してください。文字列長さは AwSr::MAX_BYTE_LENGTH（バイト）以下です。長い文字列を与えると上限を超える部分は切断します。切断で日本語文字（1 文字複数バイト）の片割れバイトができないよう注意してください。

文字列を取得／設定します。
```
const std::string& GetText() const
void  SetText(const std::string& text);
```

空文字列のオブジェクトは無効です。つぎのメソッドは空文字列だと false を返します。
```
bool IsValidData() const;
```

6.12.5 文字列の位置

グラフィックステキストの位置、外形データは下記のように持ちます。

- **最初の文字の位置 P0 の座標 (x0,y0)**
- **枠の左下隅 P1 の座標 (x1,y1)**
- **枠の右下隅 P2 の座標 (x2,y2)**
- **枠の左上隅 P3 の座標 (x3,y3)**

テキストパラメータの文字高さなどはドローイングレイアウトスペースでの大きさですが、この座標はピクチャ上の座標で、ドラフティングスケールを反映した値です。P1 → P2 の角度は文字列角度です。P1 → P3 は P1 → P2 と直交します。

文字列の位置を取得／設定するメソッドがあります。
SetLocation() メソッドは最初の文字の位置 P0 と外形を表す矩形の隅点 P1,P2,P3 を計算します。角度付き文字列の外形は傾いた矩形です。計算にはテキストパラメータと文字列を参照しますので事前に設定しておかなければなりません。

```
void SetLocation(const G2Point& location, double scale);
```

location　文字列の位置。この点 は水平方向原点基準と垂直方向原点基準の値によって、外形を表す矩形の４隅、矩形の四辺の中点、矩形の中心点のどれかと解釈します、
scale　ドラフティングスケール。

GetLocation() メソッドは文字列の位置を計算して返します。最初の文字の位置 P0 を返すのではありません。水平方向原点基準 JH、垂直方向原点基準 JV としたとき、次式で計算した点を返します。

```
   Porg = P1 + 0.5 * (JH*(P2-P1) + JV*(P3-P1))
G2Point GetLocation() const;
```

戻り値　文字列の位置。

テキストパラメータや文字列を変更したときは SetLocation() メソッドか CalcBounds() メソッドで外形を更新しなければなりません。
CalcBounds() メソッドは外形を表す矩形の隅点 P1,P2,P3 を再計算します。

```
void  CalcBounds(double scale);
```

scale　ドラフティングスケール。

文字列の外形を表す矩形の 4 点を得ます。

```
void  GetBoundPoints(G2Point points[]) const;
```

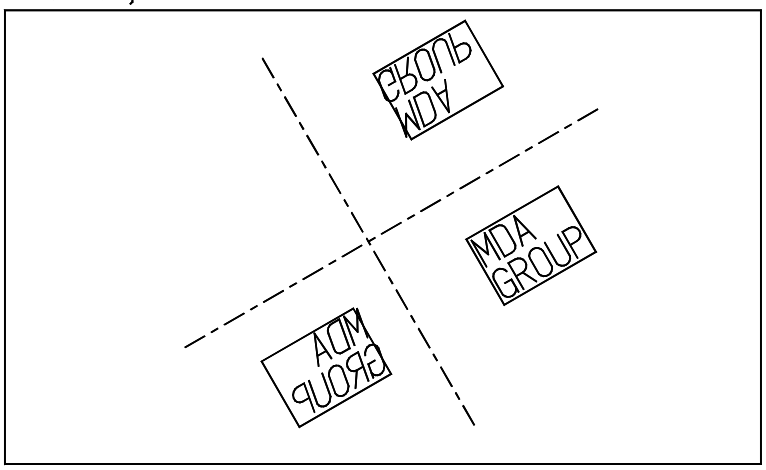
points　外形を表す矩形の 4 点を返す配列（左下、右下、右上、左上の順)。

文字列の外形を包む最小矩形を得ます。

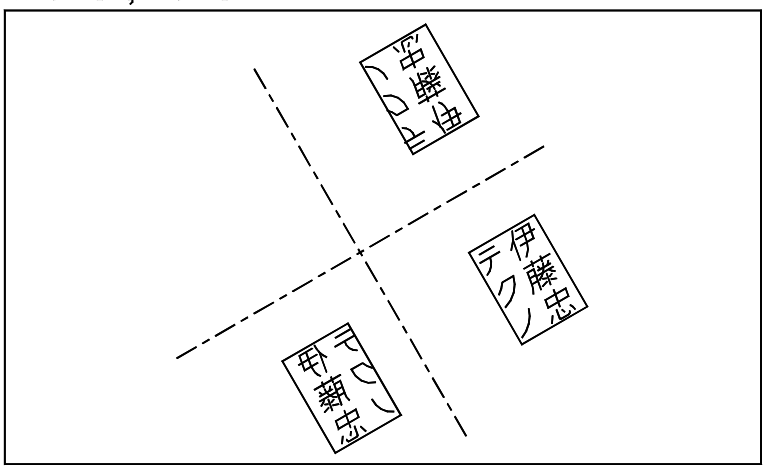
```
G2Rect GetBounds() const;
```

戻り値　外形を包む最小矩形を返します。

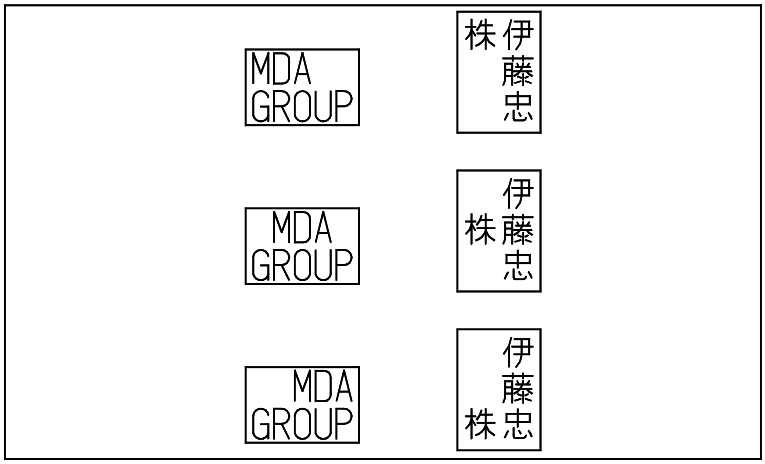
Xmirror,Ymirror



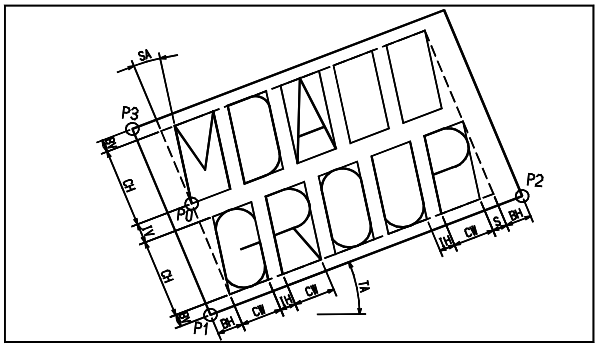
Xmirror,Ymirror



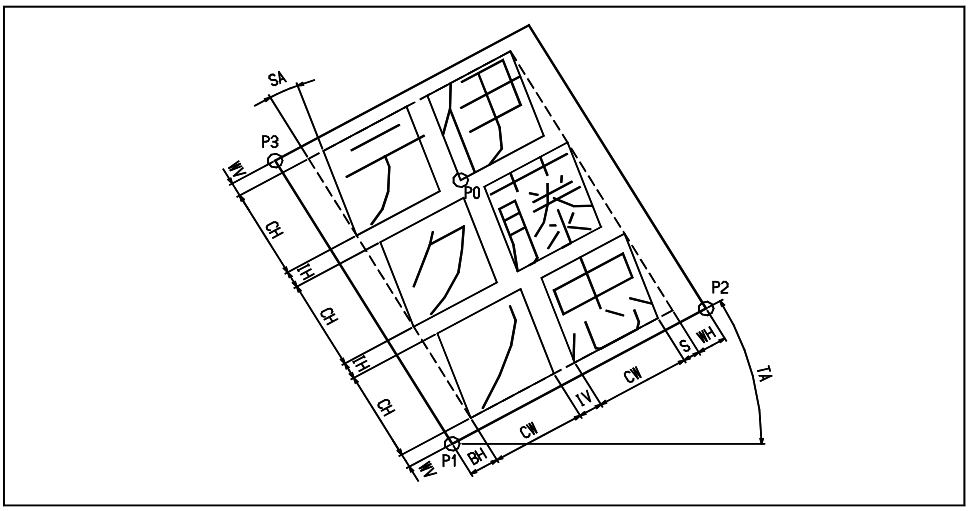
文字列表示基準 左づめ・中央・右づめ



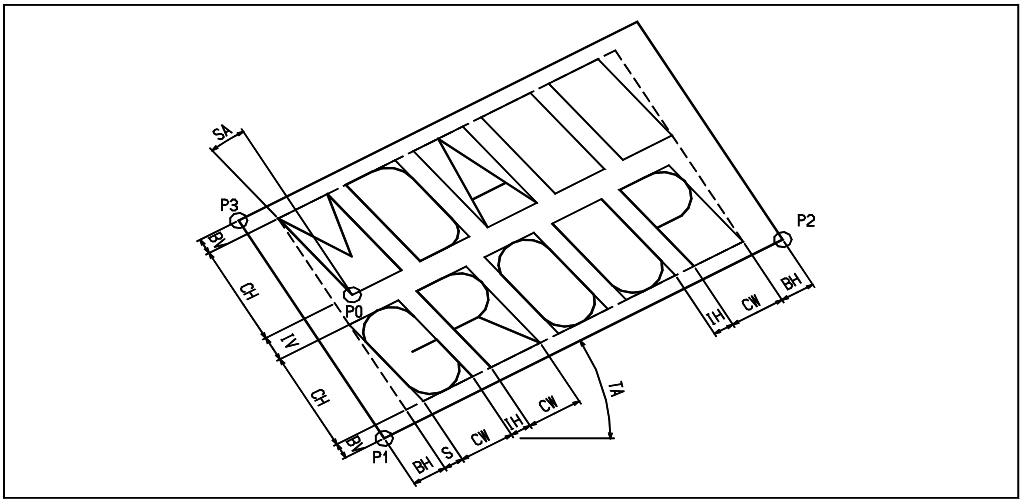
横書き〔文字傾斜角が正〕



縦書き〔文字傾斜角が正〕



横書き〔文字傾斜角が負〕



6.13 マーク AwSrMark

AwSrMark クラス はマークサブレコードを表現するクラスです。
このサブレコードはマークのパラメータ、位置を持ちます。

サブレコード線種は表示 / 非表示の制御のみに使用し、線種は常に実線で表示します。線幅は有効です。

6.13.1 コンストラクタ

```
AwSrMark();
```
マーク番号０。パラメータはなにも設定しません。

6.13.2 アクセッサ

X 座標ミラーフラッグを取得／設定します。
```
int  GetMirrorX() const;
void SetMirrorX(int flag);
```
0= しない、1= する（マークの Y 軸に対して反転する)。

Y 座標反転ミラーフラッグを取得／設定します。
```
int  GetMirrorY() const;
void SetMirrorY(int flag);
```
0= しない、1= する（マークの X 軸に対して反転する)。

マーク番号を取得／設定します。
```
int  GetMarkId() const;
void SetMarkId(int mid);
```
マーク番号 (1 - 4095)。マーク番号として 0 を設定してはなりません。マークを表示したくないが削除したくないならサブレコード線種を非表示にします。

マークサイズを取得／設定します。
```
double GetHeight() const;
void SetHeight(double height);
```
ドローイングレイアウトでのマークサイズ (0.01 - 327.76, 0.01 刻み)。

マーク角度を取得／設定します。
```
double GetAngle() const;
void SetAngle(double angle);
```
角度 (0.0 - 360.0 度)。

6.13.3 マークの位置

マークの位置、外形データは下記のように持ちます。
- **位置 P0 の座標 (x0,y0)**
- **枠の左下隅 P1 の座標 (x1,y1)**
- **枠の右下隅 P2 の座標 (x2,y2)**
- **枠の左上隅 P3 の座標 (x3,y3)**

マークサイズはドローイングレイアウトスペースでの大きさですが、この座標はピクチャ上の座標で、ドラフティングスケールを反映した値です。P1 → P2 の角度はマーク角度です。P1 → P3 は P1 → P2 と直交します。

マークの位置 P0 を取得／設定するメソッドがあります。
```
G2Point GetLocation() const;
```

```
void SetLocation(const G2Point& location);
```

CalcBounds() メソッドは外形を表す矩形の隅点 P1,P2,P3 を計算します。計算にはマークパラメータを参照しますので事前に設定しておかなければなりません。角度付きマークの外形は傾いた矩形です。またパラメータを変更したときはこのメソッドで外形を更新しなければなりません。

```
void  CalcBounds(double scale);
```

scale　　ドラフティングスケール。

マークの外形を表す矩形の 4 点を得ます。

```
void  GetBoundPoints(G2Point points[]) const;
```

points　外形を表す矩形の 4 点を返す配列（左下、右下、右上、左上の順)。

マークの外形を包む最小矩形を得ます。

```
G2Rect GetBounds() const;
```

戻り値　外形を包む最小矩形を返します。

6.14 寸法パラメータ AwSrDimension

AwSrDimension クラス は寸法サブレコードを表現するクラスです。

このサブレコードは寸法種類、寸法値の表示形式を決めるパラメータ、寸法参照点を持ちます。寸法参照点は寸法タイプごとに異なります。角度寸法以外の寸法は寸法値倍率を持ちます。実際の寸法値にこの倍率を掛けた値を寸法値とします。

寸法タイプは下記の通りです。

DIM_LINEAR	長さ寸法 (Linear Dimension)
DIM_ANGULAR	角度寸法 (Angular Dimension)
DIM_RADIUS	半径寸法 (Radius Dimension)
DIM_DIAMETER	直径寸法 (Diameter Dimension)
DIM_POINT	座標寸法 (Coordinate Dimension)
DIM_ARCLENGTH	円弧長寸法 (Arclength Dimension)
DIM_CHAMFER	面取り寸法 (Chamfer Dimension)

サブタイプは下記の通りです。これは寸法の記入方式と言ったほうがよいかもしれません。寸法タイプによっては使用できない方式があります (下記の表を参照)。

DS_SINGLE	単一寸法 (Single mode)
DS_CHAIN	直列寸法 (Chain Dimension)
DS_PARALLEL	並列寸法 (Parallel Dimension)
DS_ORDINATE	Ordinate Dimension
DS_ONESIDE	片寄せ寸法
DS_RUNNING	累進寸法 (Running Dimension)
DS_RADIALWITNESS	寸法補助線が放射状の円弧長寸法

寸法タイプ	サブタイプ
DIM_LINEAR	DS_SINGLE、DS_CHAIN、DS_PARALLEL、DS_ORDINATE、DS_ONESIDE、DS_RUNNING
DIM_ANGULAR	DS_SINGLE、DS_CHAIN、DS_PARALLEL、DS_RUNNING
DIM_RADIUS	DS_SINGLE
DIM_DIAMETER	DS_SINGLE

寸法タイプ	サブタイプ
DIM_POINT	DS_SINGLE
DIM_ARCLENGTH	DS_SINGLE、DS_RADIALWITNESS
DIM_CHAMFER	DS_SINGLE

このサブレコードは表示しませんので、サブレコード線種、線幅は意味を持ちません。

6.14.1 コンストラクタ

AwSrDimension();
　寸法タイプ。パラメータはなにも設定しません。

6.14.2 初期化

AwDrafParam オブジェクトの寸法パラメータをこのオブジェクトに一括して設定します。このメソッドは寸法参照点をクリアしますので、この後寸法参照点を設定しなければなりません。
void Init(int type, int subtype, const AwDrafParam& drafParam);
　　type　　寸法タイプ
　　subtype　　寸法サブタイプ
　　drafParam　　AwDrafParam オブジェクト

6.14.3 寸法の種類

寸法タイプを得る。
int　GetDimensionType() const;
寸法サブタイプを得る。
int　GetDimensionSubtype() const;

6.14.4 寸法値パラメータ

製図基準を取得／設定します。
int　GetDimensionTextAlignment() const;
void SetDimensionTextAlignment(int value);
　　0＝水平／寸法線分断 (ANSI)
　　1＝平行／寸法線閉 (JIS)

十進整数部の 3 桁区切り記号を取得／設定します。
int　GetThousandsSeparator() const;
void SetThousandsSeparator(int value);
　　0＝なし、1＝カンマ、2＝空白、3＝ピリオド。

十進小数点記号を取得／設定します。
int　GetDecimalSymbol() const;
void SetDecimalSymbol(int value);
　　0＝ピリオド、1＝カンマ

十進表示の小数部の「後ろのゼロを除去」を取得／設定します。
int　GetTrailZerosSuppression() const;
void SetTrailZerosSuppression(int value);
　　0＝後ろのゼロを除去する、1= 後ろのゼロも表示します。

6.14.5 長さ寸法の値パラメータ

単寸法／両寸法モードを取得／設定します。
```
int  GetDualDimensionTexts() const;
void SetDualDimensionTexts(int value);
```
0 = 単一
1 = mm/inch 併記
2 = inch/mm 併記

寸法値表示形式を取得／設定します。
```
int  GetFractionType() const;
void SetFractionType(int value);
```
0 = 十進
1 = feet & inch
2 = feet
3 = inch

寸法値単位を取得／設定します。
```
int  GetUnit() const;
void SetUnit(int value);
```
1 = milimeter、2 = centimeter、3 = meter
4 = inch、5 = feet、6 = mil (1/1000 inch)

単位記号表示を取得／設定します。
```
int  GetUnitDisplay() const;
void SetUnitDisplay(int value);
```
0 = 表示しない
1 = シンボル
2 = 省略文字

インチ分数の単位を取得／設定します。
```
int  GetFractionPrecision() const;
void SetFractionPrecision(int value);
```
寸法値表示形式が十進でないときだけ参照する。
0 = 1/1、1 = 1/2、2 = 1/4、3 = 1/8、4 = 1/16、5 = 1/32、6 = 1/64

十進表示の小数桁数を取得／設定します。
```
int  GetDecimalPrecision() const;
void SetDecimalPrecision(int value);
```
0 － 12。寸法値表示形式が十進のときだけ参照する。

半径の寸法補助記号と位置を取得／設定します。
```
int  GetRadiusSymbolPosition() const;
void SetRadiusSymbolPosition(int value);
```
0 = 前 R、1 = 後 R、2 = なし

直径の寸法補助記号と位置を取得／設定します。
```
int  GetDiameterSymbolPosition() const;
void SetDiameterSymbolPosition(int value);
```
0 = φ 前、1 = 後 φ、2 = DIA 前、3 = 後 DIA
4 = φ 前、5 = 後 φ、6 = DIA 前、7 = 後 DIA
0 ～ 3 は直径寸法と長さ寸法の φ 追加の両方に使用する。
4 ～ 7 は直径寸法に寸法補助記号を付けない。長さ寸法の φ 追加にだけ使用する。

6.14.6 角度寸法の値のパラメータ

寸法値の表示形式を取得／設定します。
```
int  GetAngleUnitDisplay() const;
void SetAngleUnitDisplay(int value);
```
0 = 度分秒 (60 進)
1 = 十進、単位表示 (Deg)
2 = 十進、単位表示なし
3 = 十進、単位表示 (° シンボル)

度分秒 (60 進) で表示する最小単位を取得／設定します。
```
int  GetAngleUnit() const;
void SetAngleUnit(int value);
```
1 = 度まで、2 = 分まで、3 = 秒まで。

十進表示の小数桁数を取得／設定します。
```
int  GetAngleDecimalPrecision() const;
void SetAngleDecimalPrecision(int value);
```
小数桁数は 0 - 12。

外側寸法線のスタイルを取得／設定します。
```
int  GetExteriorArrowLineStyle() const;
void SetExteriorArrowLineStyle(int value);
```
0 = 線分、1 = 円弧。

6.14.7 寸法許容差のパラメータ

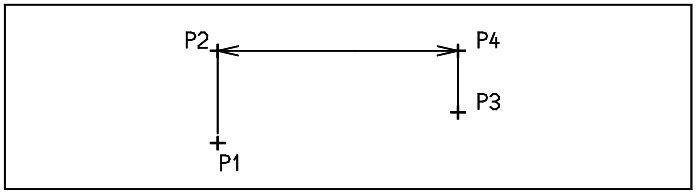
十進表示の小数桁数を取得／設定します。
```
int  GetToleranceDecimalPrecision() const;
void SetToleranceDecimalPrecision(int value);
```
小数桁数は 1 - 12。

十進表示の小数部の「後ろのゼロを除去」を取得／設定します。
```
int  GetToleranceTrailZerosSuppression() const;
void SetToleranceTrailZerosSuppression(int value);
```
0 = 後ろのゼロを除去する、1= 後ろのゼロも表示する。

6.14.8 長さ寸法

長さ寸法は図の P1, P2, P3, P4 の 4 つの参照点を持ちます。
P1 → P2, P3 → P4 は寸法補助線で平行でなければなりません。
P2 → P4 は寸法線がのる直線、寸法値は P2,P4 の距離です。

長さ寸法　分類コード（1）



次のメソッドは長さ寸法の参照点を設定するメソッドです。

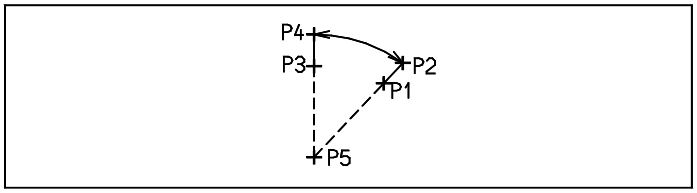
```
bool SetDimLinear(const G2Point& ps1, const G2Point& pe1,
                  const G2Point& ps2, const G2Point& pe2,
                  double scale);
```

ps1　　寸法記入点 1(図の P1)
pe1　　寸法線 1 ／寸法補助線 1 の端点 (図の P2)
ps2　　寸法記入点 2(図の P3)
pe2　　寸法線 2 ／寸法補助線 2 の端点 (図の P4)
scale　寸法値の倍率 (0 < scale)
戻り値　成功したとき true を返します。

6.14.9 角度寸法

角度寸法は図の P1, P2, P3, P4, P5 の５つの参照点を持ちます。
P1 → P2, P3 → P4 は寸法補助線、P5 は中心点です。
１）寸法補助線は中心点を通過する直線上になければならず、中心点を含んではなりません。
２）P2, P4 は P5 を中心とする 円上になければなりません（寸法線がのる円)。
寸法値の角度は P2 から P4 に向かって反時計回りに測定します。

角度寸法　分類コード〔2〕



次のメソッドは角度寸法の参照点を設定するメソッドです。

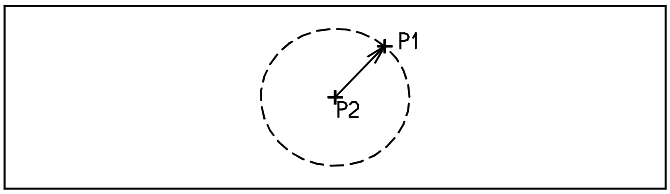
```
bool SetDimAngular(const G2Point& ps1, const G2Point& pe1,
                   const G2Point& ps2, const G2Point& pe2,
                   const G2Point& pc);
```

ps1　　寸法記入点 1(図の P1)
pe1　　寸法線 1 ／寸法補助線 1 の端点 (図の P2)
ps2　　寸法記入点 2(図の P3)
pe2　　寸法線 2 ／寸法補助線 2 の端点 (図の P4)
pc　　中心点 (図の P5)
戻り値　成功したとき true を返します。

6.14.10 半径寸法

半径寸法は図の P1, P2 の２つの参照点を持ちます。
P1　円周上の点
P2　円中心点

半径寸法　分類コード〔3〕

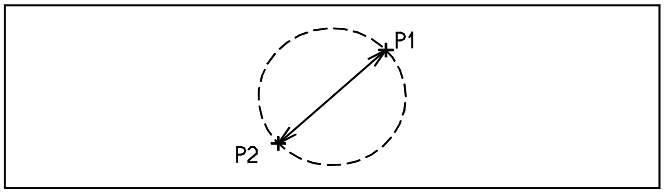


次のメソッドは半径寸法の参照点を設定するメソッドです。
```
bool SetDimRadius(const G2Point& pcen, const G2Point& pcir, double scale);
```
pcen　　円中心 (図の P2)
pcir　　円周点 (図の P1)
scale　　寸法値の倍率 (0 < scale)
戻り値　成功したとき true を返します。

6.14.11 直径寸法

直径寸法は図の P1, P2 の２つの参照点を持ちます。
P1, P2 は円周上の対向点です（線分 P1 → P2 は円中心点を通過する)。

直径寸法　分類コード〔4〕

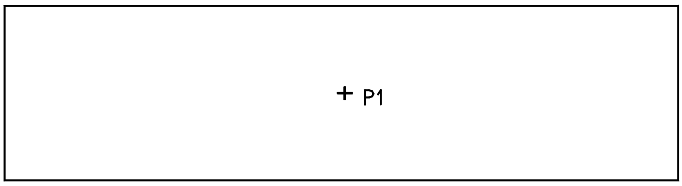


次のメソッドは直径寸法の参照点を設定するメソッドです。
```
bool SetDimDiameter(const G2Point& pcen, const G2Point& pcir, double scale);
```
pcen　　円中心
pcir　　円周点 (図の P1)
scale　　寸法値の倍率 (0 < scale)
戻り値　成功したとき true を返します。

6.14.12 座標寸法

座標寸法は図の P1 の参照点だけを持ちます。

点寸法　分類コード〔5〕



次のメソッドは座標寸法の参照点を設定するメソッドです。

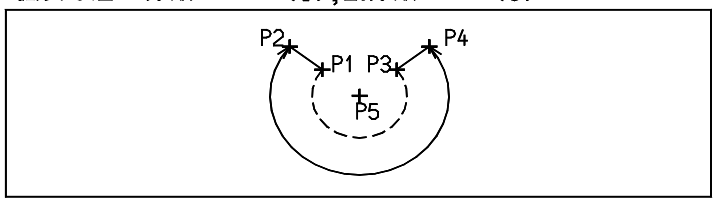
```
bool SetDimPoint(const G2Point& point, double scale);
```

point　寸法記入点 (図の P1)
scale　寸法値の倍率 (0 < scale)
戻り値　成功したとき true を返します。

6.14.13 円弧長寸法

円弧長寸法は図の P1, P2, P3, P4, P5 の 5 つの参照点を持ちます。
P1, P3 は参照円上の点、P5 は参照円の中心点です。参照円は P1 から P3 へ向かって反時計回りとします。
P1 → P2、P3 → P4 は寸法補助線です。P2,、P4 は P5 を中心とする 円上になければなりません（寸法線がのる円)。
図は寸法補助線は（角度寸法のように）放射状にするか、(長さ寸法のように）平行にすることができます。図は寸法補助線が中心点を通過する放射状形式を示しています。平行線形式は参照円弧の角度が 180 度未満でなければならない制限があります。

弧長寸法　分類コード〔6),副分類コード〔0〕



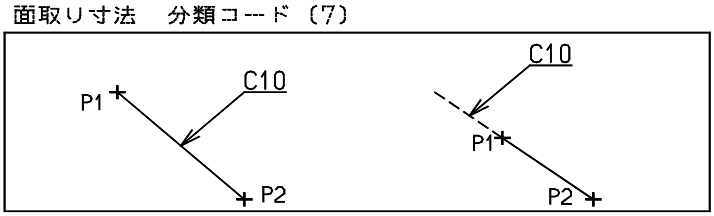
次のメソッドは円弧長寸法の参照点を設定するメソッドです。

```
bool SetDimArclength(const G2Point& ps1, const G2Point& pe1,
                     const G2Point& ps2, const G2Point& pe2,
                     const G2Point& pc, double scale);
```

ps1　寸法記入点 1(図の P1)
pe1　寸法線 1 ／寸法補助線 1 の端点 (図の P2)
ps2　寸法記入点 2(図の P3)
pe2　寸法線 2 ／寸法補助線 2 の端点 (図の P4)
pc　円中心点 (図の P5)
scale　寸法値の倍率 (0 < scale)
戻り値　成功したとき true を返します。

6.14.14 面取り寸法

面取り寸法は図の P1, P2 の 2 つの参照点を持ちます。
P1 → P2 を 45° 面取り線の端点とみなし、寸法値は P1 → P2 の距離から計算します。



次のメソッドは面取り寸法の参照点を設定するメソッドです。
```
bool SetDimChamfer(const G2Line& line, double scale);
```
ps1　　面取り線 (図の P1->P2)
scale　　寸法値の倍率 (0 < scale)
戻り値　成功したとき true を返します。

6.14.15 参照点の取得

寸法値の倍率を得ます。
```
double GetScale() const;
```
戻り値　寸法値の倍率を返します。角度寸法は対象外ですが 1.0 を返します。

寸法参照点を得ます。
```
int  GetReferencePoints(G2Point points[]) const;
```
points　寸法参照点を配列を格納する配列。
戻り値　寸法参照点数を返します。点数は寸法タイプに依存します。

6.15 幾何公差枠 AwSrGtFrame

AwSrGtFrame クラス は幾何公差サブレコードを表現するクラスです。

このサブレコードは幾何公差の位置、外形を持ちます。
- **中心点 P0 の座標 (x0,y0)**
- **枠の左下隅 P1 の座標 (x1,y1)**
- **枠の右下隅 P2 の座標 (x2,y2)**
- **枠の左上隅 P3 の座標 (x3,y3)**

このサブレコードは表示しませんので、サブレコード線種、線幅は意味を持ちません。

6.15.1 コンストラクタ

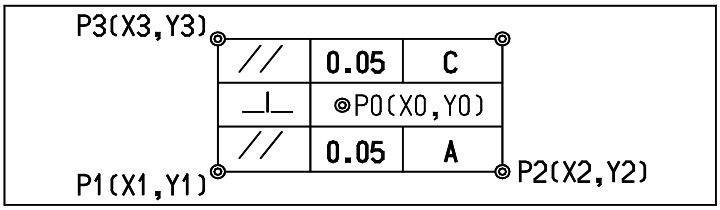
```
AwSrGtFrame();
```

6.15.2 アクセッサ

幾何公差枠の位置、外形を設定します。
```
void Set(double width, double height, double angle, const G2Point& center);
```
width	外形の幅 (0 < width)
height	外形の高さ (0 <height)
angle	底辺の角度（単位は度）
center	枠の位置（中心点）

幾何公差枠の中心点を取得します。
```
G2Point GetOrigin() const;
```
戻り値　中心点を返します。

幾何公差枠の外形を表す矩形の 4 点を得ます。
```
void  GetBoundPoints(G2Point points[]) const;
```
points　外形を表す矩形の 4 点を返す配列（左下、右下、右上、左上の順)。



6.16 塗り潰しパラメータ AwSrAflParam

AwSrAflParam クラス は塗り潰しパラメータサブレコードを表現するクラスです。

このサブレコードは塗り潰しパターン、横幅、縦幅、角度、格子基準点、格子の角度と間隔を持ちます。
SXF の仕様を取り込み拡張しています。パターンを並べる格子は直交な格子でしたが、斜交する格子も可能にするため格子軸の角度と間隔を 2 組持ちます。パターンは格子に指定した横幅、縦幅、角度で配置します。
しかし SXF 以外は、従来と同じになるよう以下の制約を守っています。
- パターン配置角度＝格子の軸 #1 の角度。
- 格子の軸 #1 と格子の軸 #2 は直交（軸 #2 の角度 = 軸 #1 の角度 + 90 度)。
- パターン大きさ #1（横幅）<= 格子の間隔 #1。
- パターン大きさ #2（縦幅）<= 格子の間隔 #2。

サブレコードの線種は表示／非表示の制御にのみ使用します。サブレコード線幅は意味を持ちません。

6.16.1 コンストラクタ

```
AwSrAflParam();
```

6.16.2 パターン

塗り潰しのパターンは 3 種類あります。

PATTERN_DEVICE　機器（ディスプレイ、プリンタなど）固有のパターン。
PATTERN_MARK　マーク。
PATTERN_CHAR　文字。

塗り潰しパターンの種類を得る。
```
int GetPatternType() const;
```

１）機器固有のパターンを使うとき
機器固有のパターン番号を取得／設定します。
```
void  SetDevicePattern(int npat);
int   GetPattern() const;
```
実際には「ベタ塗り」だけを使用しますので npat = 0 です。

２）パターンとしてマークを使うとき
パターンとして使用するマーク番号を取得／設定します。
```
void  SetPattern(int mark);
int   GetPattern() const;
```
マーク番号 (1-4095)。

３）パターンとして文字を使うとき。
パターンとして使用する文字を設定します。文字は 1 文字であり文字列ではありません。
```
void SetPattern(int asciiFont, int japaneseFont, const std::string& string);
```
asciiFont　ASCII 文字のフォント番号。
japaneseFont　日本語文字のフォント番号。
string　ASCII 文字または日本語文字（2 バイト、日本語 EUC)。
2 つのテキストフォント番号を渡しますが、文字が ASCII 文字か日本語文字かに応じて一方だけを使います。

パターン文字を得ます。
```
int GetPattern(std::string* string) const;
```
string　ASCII 文字または日本語文字（2 バイト）を設定します。
戻り値　文字の種類を返します。
AwEucEncoder::TYPE_PRINTABLE　ASCII 文字。
AwEucEncoder::TYPE_MULTIBYTE　日本語文字。
0 はパターン文字はなし。

テキストフォント番号を得ます。
```
int GetTextFont() const;
```
戻り値　テキストフォント番号を返します。

6.16.3 パターン格子

格子の基準点を取得／設定します。
```
G2Point GetThroughPoint() const;
void SetThroughPoint(const G2Point& point);
```

格子の軸角度を取得／設定します。
```
double GetVectorAngle1() const;
void SetVectorAngle1(double angle);
double GetVectorAngle2() const;
void SetVectorAngle2(double angle);
```
格子の軸の角度（0.0 - 360.0 度)。

格子間隔を取得／設定します。
```
double GetVectorLength1() const;
void SetVectorLength1(double length);
double GetVectorLength2() const;
```

```
void SetVectorLength2(double length);
```
格子の間隔（0 ＜ length)。

パターンの配置角度を取得／設定します。
```
int  GetPatternType() const;
void SetAngle(double angle);
```
角度 (0.0-360.0 度)。

パターンの幅と高さを取得／設定します。
```
double GetWidth() const;
void   SetWidth(double width);
double GetHeight() const;
void   SetHeight(double height);
```
幅と高さはゼロでない値です。負の値はパターンを反転させます。負の幅は左右反転し、負の高さは上下反転します。

6.17 ハッチングパラメータ AwSrXhtParam

AwSrXhtParam クラス は塗り潰しパラメータサブレコードを表現するクラスです。

このサブレコードはハッチングパターンを持ちます。SXF の仕様を取り込み拡張しています。ここでは SXF 仕様の説明は行いません。

6.17.1 コンストラクタ

```
AwSrXhtParam();
```

6.17.2 アクセッサ

ハッチング仕様を得ます。
```
int GetPatternType() const;
```
0 = Advance CAD 仕様 , 1 = SXF 仕様。

ハッチングパターン番号を得ます。
```
int GetPatternId() const;
```
0　　= パターンを使用しないで表示 / 非表示線の数を使う。
1 - 32 = パターン番号。

6.17.3 パターン番号を使用しない表示

表示線の数と非表示線の数を使ってハッチングパターンを直接指示する方法です。
たとえば、次のような設定が考えられます。
表示する線数 1、表示しない線数 0 は最も単純で { 表示} を繰り返します。
表示する線数 1、表示しない線数 1 は { 表示、非表示} を繰り返します。
表示する線数 2、表示しない線数 1 は { 表示、表示、非表示} を繰り返します。
表示する線の数は 1 以上 (1-1000)、表示しない線の数は 0 以上 (0-1000) でなければなりません。

ハッチング線の表示はアイテムの線種、線幅を使います。サブレコード線種、線幅が設定してあればそれが優先します。

ハッチングのスタイル（平行かクロスか）を選べます。

表示線の数と非表示線の数を設定します。

```
void  SetLinePattern(int visible, int invisible);
```

表示線の数と非表示線の数を得ます。
```
int   GetVisibleCount() const;
int   GetInvisibleCount() const;
```

ハッチングのスタイルを取得／設定します。
```
int   GetMode() const;
void  SetMode(int mode);
```
0= 平行、1= クロス

6.17.4 ハッチング線パラメータ

以下のメソッドの引数 index は 0 とします。

ハッチング線の角度を取得／設定します。
```
double GetAngleAt(int index) const;
void   SetAngleAt(int index, double angle);
```
線の角度 (0.0 - 360.0 度)。

ハッチング線の間隔を取得／設定します。
```
double GetIntervalAt(int index) const;
void   SetIntervalAt(int index, double interval);
```
線の間隔 (ドローイングレイアウトスペース、0 <interval)。

ハッチング線の通過点を取得／設定します。
```
G2Point GetThroughPointAt(int index) const;
void SetThroughPointAt(int index, const G2Point& point);
```
線の通過点 (モデルスペース)。

6.18 日付 AwSrTimestamp

AwSrTimestamp クラス は日付サブレコードを表現するクラスです。

このサブレコードは日付を持ちます。シンボルアイテム、サブモデルアイテムの作成日付として使用します。サブモデルアイテムと元のモデルファイルの日付を比較して更新するときに必要です。

このサブレコードは表示しませんので、サブレコード線種、線幅は意味を持ちません。

6.18.1 コンストラクタ

```
AwSrTimestamp();
```

6.18.2 アクセッサ

日付を取得／設定します。
```
void GetDate(short date[], short time[]) const;
bool SetDate(const short date[], const short time[]);
```
date[0] : 日 (1-31)
date[1] : 月 (1-12)
date[2] : 西暦年 (1986 - 2076)
time[0] : 時 (0-23)
time[1] : 分 (0-59)

time[2] : 秒 (0-59)

次のメソッドは現在の日付を設定します。
```
void SetCurrentDate();
```

次のメソッドは日付の文字列を得ます。
```
std::string GetDateString() const;
```
戻り値　日付の文字列を返します。”yyyy/mm/dd hh:mm:ss” の形式の文字列を得ます。長さは 19 文字で、dd と hh の間に空白文字があります。

6.18.3 比較

日付を比較するには次のメソッドを使います。
```
int Compare(const AwSrTimestamp& date) const;
```
戻り値　比較結果を返します。
- -1　このオブジェクトの日付 < 引数の日付。
- 0　ふたつの日付が等しい。
- 1　このオブジェクトの日付 > 引数の日付。

日付がある期間内にあるか判定するには次のメソッドを使います。
```
bool InSpan(const AwSrTimestamp& start, const AwSrTimestamp& end) const;
```
戻り値　start の日付 <= このオブジェクトの日付 <= end の日付 なら true を返します。

6.19 配置パラメータ AwSrPlacement

AwSrPlacement クラス は配置パラメータサブレコードを表現するクラスです。

このサブレコードはシンボルアイテム、サブモデルアイテム、ＡＰＧ アイテムの配置情報を持ちます。
1. 配置点 P0 の座標 (x0,y0)
2. 枠の左下隅 P1 の座標 (x1,y1)
3. 枠の右下隅 P2 の座標 (x2,y2)
4. 枠の左上隅 P3 の座標 (x3,y3)
5. シンボル／サブモデル配置時の X 方向縮尺値。
   縮尺値　＞　0　反転しない
   縮尺値　＜　0　左右反転する（X 座標を反転）
6. シンボル／サブモデル配置時の Y 方向縮尺値。
   縮尺値　＞　0　反転しない
   縮尺値　＜　0　上下反転する（Y 座標を反転）
7. シンボル／サブモデル配置時の回転角度。
   単位は度 (-360 ＜回転角度＜ 360)。P1 → P2 の角度（モデル座標系 X 軸からの角度）と一致しなければならない。

このサブレコードは表示しませんので、サブレコード線種、線幅は意味を持ちません。

6.19.1 コンストラクタ

```
AwSrPlacement(const G2Point& origin, const double scale[], double angle,
              const G2Rect& rect, bool bTrans);
```
- origin　配置点。
- scale　縮尺値（X-scale, Y-scale)。
- angle　配置角度。単位は度。
- rect　外形を与える矩形。

bTrans　true の時は rect で与えられた外形に縮尺 (scale)、回転 (angle)、移動 (origin) を適用した結果を使って外形 (P1, P2, P3) を設定する。

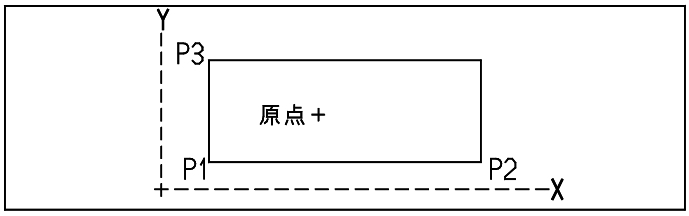
6.19.2 アクセッサ

配置点を得ます。
```
G2Point GetLocation() const;
```

外形を表す矩形の 4 点を得ます（左下、右下、右上、左上の順)。
```
void GetBoundPoints(G2Point points[]) const;
```

配置角度を得ます。
```
double GetAngle() const;
```

配置縮尺値（X-scale, Y-scale）を得ます。
```
void GetScale(double scale[]) const;
```



6.20 元のアイテム属性 AwSrAttributes

AwSrAttributes クラス は元のアイテム属性サブレコードを表現するクラスです。

このサブレコードは複合アイテム、サブモデル、シンボルなどの構造化アイテムで、それらのアイテムを作る元になったアイテムの属性を保持するのに用います。構造化アイテムの表示の制御やアイテムを分解して元になったアイテムに戻すなどに使います。
このサブレコードの後に元のアイテムのサブレコードを置きます。その終了は次の元のアイテム属性サブレコードまたはアイテムの終了サブレコード (AwSr::EOI) の直前のサブレコードです。

- (省略)
- サブレコード (AwSr::ATTRIBUTS2)
- 元のアイテムのサブレコードの並び
- サブレコード (AwSr::ATTRIBUTS2)
- 元のアイテムのサブレコードの並び
- (省略)

この説明は単純化しているので、複合アイテムのようなアイテムの階層を持たないアイテムに当てはまります。サブモデルがシンボルを含んでいると 2 階層になりますから、サブレコード AwSr::ATTRIBUTES2 のマッチングはアイテムの階層を考慮しなくてはなりません。

このサブレコードはアイテムタイプ、クラス、レビジョン、線種、線幅、ブランクフラグ、色番号を持ちます。

6.20.1 コンストラクタ

```
AwSrAttributes();
AwSrAttributes(int type, int iclass, int revision, int font = 1, int weight = 1,
               int color = 1, int blank = 0);
AwSrAttributes(const AwItemAttributes& attr);
```

6.20.2 アクセッサ

アイテムタイプを取得／設定します。
```
int GetItemType() const;
void SetItemType(int value);
```

クラスを取得／設定します。
```
int GetItemClass() const;
void SetItemClass(int value);
```

レビジョンを取得／設定します。
```
int GetItemRevision() const;
void SetItemRevision(int value);
```

線種を取得／設定します。
```
int GetItemLineFont() const;
void SetItemLineFont(int value);
```

線幅を取得／設定します。
```
int GetItemLineWeight() const;
void SetItemLineWeight(int value);
```

ブランクフラグを取得／設定します。
```
int GetItemBlankFlag() const;
void SetItemBlankFlag(int value);
```

色番号を取得／設定します。
```
int GetItemColor() const;
void SetItemColor(int value);
```

アイテム属性を取得します
```
void GetAttributes(AwItemAttributes* pAttr) const;
```
pAttr　AwItemAttributes オブジェクトに、このサブレコードが持つアイテム属性を設定します。ピクチャ番号は変えません。

6.21 アソシエイション AwSrAssociation

AwSrAssociation クラス はアソシエイションサブレコードを表現するクラスです。

このサブレコードは関係するアイテムを記憶します。
1. 関係の区分
   メジャーコードとマイナーコードの 2 つの整数で「関係」を区別します。
   メジャーコードは 1-255 の値で 0 は使用しません。
   マイナーコードは 0-255 の値でメジャーコードに依存します。
2. 相手のアイテム識別子（正整数）

アイテム A と B の間に関係 X を付けるとすると次のようにします。
アイテム A：メジャーコード X, マイナコード 1、アイテム識別子 B。

アイテム B：メジャーコード X, マイナコード 2、アイテム識別子 A。

このサブレコードは表示しませんので、サブレコード線種、線幅は意味を持ちません。

6.21.1 コンストラクタ

```
AwSrAssociation();
```

6.21.2 アクセッサ

関係の区分を取得／設定します。

```
int  GetMajorCode() const;
int  GetMinorCode() const;
void SetCategory(int major, int minor);
```

相手のアイテム識別子を取得／設定します。

```
int  GetItemId() const;
```

戻り値　アイテム識別子。

```
void SetItemId(int id, bool bAny = false);
```

id　アイテム識別子。

bAny　負のアイテム識別子を設定するときに true とします。ゼロや負のアイテム識別子を設定してはなりませんので、この引数は常に false とします。

6.22 アイテムの終端 AwSrEoi

AwSrEio クラス は終了サブレコードを表現するクラスです。

このサブレコードはアイテムのデータの最後であることを表します。アイテムのサブレコード並びの最後にこのサブレコードを置き、終了であることを明示します。このサブレコードを持たないアイテムは不正なアイテムです。またこのサブレコード以後にサブレコードを持っているアイテムも不正なアイテムです。
このサブレコードは表示しませんので、サブレコード線種、線幅は意味を持ちません。

6.22.1 コンストラクタ

```
AwSrEio();
```

第 7 章　メッセージなど

7.1　コマンド識別番号

● 関数一覧

関数名	機能
cmdidget	コマンド識別番号を得る。
cmdidset	コマンド識別番号を設定する。
cmdidcla	コマンド識別番号をクリアする。
cmdmdfget	修飾子識別番号を得る。
cmdmdfset	修飾子識別番号を設定する。
cmdmdfcla	修飾子識別番号をクリアする。
ldispatch	ディスパッチャ番号を得る。
ldriver	ドライバ番号を得る。
lformat	フォーム番号を得る。

7.1.1　コマンド識別番号の取得

指定したコマンドカテゴリのコマンド識別番号を得る。

【呼出し形式】

```
void cmdidget(int cmdctg, int *disp, int *driv, int *form)
```

【入力引数】

```
cmdctg    コマンドカテゴリ。(1, 3, 4, 5)
```

【出力引数】

```
disp      ディスパッチャ番号
driv      ドライバ番号
form      フォーム番号
```

7.1.2　コマンド識別番号の設定

指定したコマンドカテゴリのコマンド識別番号を設定する。

【呼出し形式】

```
void cmdidset(int cmdctg, int disp, int driv, int form)
```

【入力引数】

cmdctg	コマンドカテゴリ。(1, 3, 4, 5)
disp	ディスパッチャ番号
driv	ドライバ番号
form	フォーム番号

7.1.3 コマンド識別番号のクリア

指定したコマンドカテゴリのコマンド識別番号をクリアする。

【呼出し形式】

```
void cmdidcla(int cmdctg)
```

【入力引数】

cmdctg	コマンドカテゴリ。(1, 3, 4, 5)

7.1.4 修飾子識別番号の取得

指定したコマンドカテゴリの修飾子識別番号を得る。

【呼出し形式】

```
void cmdmdfget(int cmdctg, int *disp, int *driv, int *form)
```

【入力引数】

cmdctg	コマンドカテゴリ。(1, 3, 4, 5)

【出力引数】

disp	ディスパッチャ番号
driv	ドライバ番号
form	フォーム番号

7.1.5 修飾子識別番号の設定

指定したコマンドカテゴリの修飾子識別番号を設定する。

【呼出し形式】

```
void cmdmdfset(int cmdctg, int disp, int driv, int form)
```

【入力引数】

cmdctg	コマンドカテゴリ。(1, 3, 4, 5)
disp	ディスパッチャ番号
driv	ドライバ番号
form	フォーム番号

7.1.6 修飾子識別番号のクリア

指定したコマンドカテゴリの修飾子識別番号をクリアする。

【呼出し形式】

```
void cmdmdfcla(int cmdctg)
```

【入力引数】

cmdctg　コマンドカテゴリ。(1, 3, 4, 5)

7.1.7 ディスパッチャ番号の取得

コマンドまたは修飾子のディスパッチャ番号を得る。

【呼出し形式】

```
int ldispatch(int cmdctg)
```

【入力引数】

cmdctg　コマンドカテゴリ。(1, 2, 3, 4, 5)
1, 3, 4, 5　：　このコマンドカテゴリのコマンドのディスパッチャ番号を得る
2　：　アクティブコマンドの修飾子のディスパッチャ番号を得る

【戻り値】

ディスパッチャ番号

7.1.8 ドライバ番号の取得

コマンドまたは修飾子のドライバ番号を得る。

【呼出し形式】

```
int ldriver(int cmdctg)
```

【入力引数】

cmdctg　コマンドカテゴリ。(1, 2, 3, 4, 5)
1, 3, 4, 5　：　このコマンドカテゴリのコマンドのドライバ番号を得る
2　：　アクティブコマンドの修飾子のドライバ番号を得る

【戻り値】

ドライバ番号

7.1.9 フォーム番号の取得

コマンドまたは修飾子のフォーム番号を得る。

【呼出し形式】

```
int lformat(int cmdctg)
```

【入力引数】

cmdctg　コマンドカテゴリ。(1, 2, 3, 4, 5)
1, 3, 4, 5　：　このコマンドカテゴリのコマンドのフォーム番号を得る
2　：　アクティブコマンドの修飾子のフォーム番号を得る

【戻り値】

フォーム番号

7.2 トークン

関数名	機能
tknclear	構造体 TOKEN のメンバー変数を初期化する。
tknmsk001	入力可能なトークンの種類を設定する。
Tpntinput	直前のトークンがテンポラリポイントコマンドか判定する。

7.2.1 構造体 TOKEN の初期化

構造体 TOKEN のメンバ変数を初期化する。

【呼出し形式】

```
void tknclear(TOKEN *token)
```

【出力引数】

token　初期化する TOKEN 構造体。
各メンバ変数は以下の値になる。

typ	:	TknVOID
cid	:	cid[0] ～ [4] まですべて 0
pnt	:	<0.0, 0.0>
xflg	:	1
yflg	:	1
input	:	<0.0, 0.0>
scl	:	0.0
sclflg	:	0
txt	:	'¥0'
vp	:	アクティブビューポート番号

7.2.2 入力可能なトークンの設定

入力可能なトークンの種類を設定する。

【呼出し形式】

```
void tknmsk001(int ictg, int mode, int type)
```

【入力引数】

ictg　コマンドカテゴリ。(1, 3, 4, 5)

mode　初期設定するか追加するかのスイッチ

-1	:標準設定に戻す
0	:初期設定する
1	:追加する

type　トークンタイプ。mode が 0 または 1 の時に参照する

TknCMD	:	コマンド
TknMDF	:	修飾子
TknCOD	:	座標
TknDIG	:	デジタイズ座標
TknSCL	:	数値

TknTXT : 文字列
TknIDP : アイテム識別子
TknBSP : バックスペース
TknSPC : スペースキー
TknVEC : ベクトル
TknMZN : メッセージゾーン
TknGZN : グラフィックゾーン
TknUZN : テンポラリウィンドウ
TknEOC : CE

注意

(1) トークンタイプ TknCMD, TknMDF, TknCOD, TknDIG, TknSCL, TknTXT, TknIDP, TknBSP, TknSPC, TknVEC, TknMZN, TknEOC はこの関数が呼ばれない時はいつでも有効になっている。(標準状態)
(2) この関数での設定は次の一回のトークンの入力だけに有効である。トークンが入力されるとこの設定はクリアされる。(標準状態に戻る)
(3) トークンタイプ TknCOD, TknDIG, TknIDP, TknVEC はグループ化されており、このグループ内のどれかを有効にするとこのグループ全てが有効になる。例えばトークンタイプ TknCOD を有効にすると、トークンタイプ TknDIG, TknIDP, TknVEC も有効になる。
(4) トークンタイプ TknBSP, TknEOC はいかなる場合も有効になっている。たとえばトークンタイプ TknSCL だけを有効と指定した場合は、トークンタイプ TknSCL, TknBSP, TknEOC が有効になる。
(5) トークンタイプ TknCOD, TknDIG, TknIDP, TknVEC と TknGZN は排他的な関係にあるので、同時に有効にはできない。例えばトークンタイプ TknGZN を有効にすると、TknCOD, TknDIG, TknIDP, TknVEC は入力不可になる。
(6) トークンタイプ TknCMD を有効にすると、トークンタイプ TknMDF, TknMZN も有効になる。

例.

1. トークンタイプ TknSPC と TknGZN だけを有効にする。

```
tknmsk001(1, 0, TknSPC); /* TknSPC だけを有効にする */
tknmsk001(1, 1, TknGZN); /* さらに TknGZN も有効にする */
```

2. 標準状態にトークンタイプ TknUZN も有効にする。

```
tknmsk001(1, 1 , TknUZN);
```

7.2.3 ポイントコマンドトークンか判定

直前の入力トークンがテンポラリポイントコマンドトークンか判定します。コマンド構文上でテンポラリポイントとアイテム選択が競合する場合があります。そのような場合に、ディジタイズトークンがテンポラリポイントコマンドの引数なのかどうか判定するのに使います。

【呼出し形式】

```
void Tpntinput()
```

【戻り値】

1 : テンポラリポイントコマンドである。
0 : テンポラリポイントコマンドではない。

7.3 メッセージ

(1) ゾーンはメニューのゾーン定義ファイル (ACADZON.MEN) で指定している。

(2) ゾーンは表形式 (行列) に区分けしてあり、メッセージ開始位置を指定できる。

ゾーン	行	列
#6 メッセージゾーン	1-3	1-3

ゾーン	行	列
#7 プロンプトゾーン	1-3	1-3

(3) メッセージゾーンは、コマンドの状態表示に使用する。

(4) プロンプトゾーンは、操作促進メッセージ行 (#1 と #2) とエラーメッセージ行 (#3) の２つから成る。
行 #1 と #2 は、一般のメッセージ表示には使用できない。
エラーメッセージと同様に、すぐに消えてもよいメッセージには行 #3 を使用できる。

(5) メッセージ番号に対応するメッセージがなければ表示しない。

● 関数一覧

関数名	機能
Errorb	ベルを鳴らす。
Errorcode	エラーメッセージを表示する。
Mesagdisp	メッセージを表示する。
Mesageras	メッセージを消す。
Opmsgcode	操作促進メッセージを表示する。
Menuzone	ゾーンの情報を得る。

7.3.1 エラーベル

ベルを鳴らす。

【呼出し形式】

```
void Errorb(void)
```

7.3.2 エラーメッセージの表示

エラーメッセージを表示する。プロンプトゾーンの 3 行目に表示する。

【呼出し形式】

```
void Errorcode(int msgid)
```

【入力引数】

```
msgid   メッセージ番号
        msgid ＞ 0  ベルも鳴らす
        msgid ＜ 0  ベルは鳴らさない
```

7.3.3 メッセージの表示

メッセージを表示する。標準では MZONECOLOR1 は緑色、MZONECOLOR2 は水色に設定されている。コマンドオプションとして選択可能なメッセージは MZONECOLOR2 を、単なるメッセージは MZONECOLOR1 を使用している。詳しくは『システム管理者の手引き』「８章メニューの作成」を参照のこと。

【呼出し形式】

```
void Mesagdisp(int iz, int row, int col, int msgid, int type, const void* data)
```

【入力引数】

iz	メッセージを表示するゾーンとカラー指定 MZONECOLOR1 : メッセージゾーンにカラー１で表示する MZONECOLOR2 : メッセージゾーンにカラー２で表示する INPZONE : プロンプトゾーンに表示する
row	行番号
col	列番号
msgid	メッセージ番号。0 : メッセージなし
type	メッセージの後に表示するデータのタイプ。 0 :なし, 1 : short (16 bits), 2 : float, 3 : double 4 :int (32 bits), 10 * 文字数 : 文字列
data	メッセージの後に表示するデータ。

7.3.4 メッセージの消去

メッセージを消す。

【呼出し形式】

```
void Mesageras(int iz, int row, int col)
```

【入力引数】

iz	ゾーン番号 MSGZONE : メッセージゾーン INPZONE : プロンプトゾーン
row	行番号。0 : すべての行
col	列番号。0 : すべての列

7.3.5 操作促進メッセージ表示

操作促進メッセージを表示する。プライマリメッセージが指定されるとセカンダリメッセージはクリアされる。従って操作促進メッセージを２行表示したい場合は、プライマリメッセージを指定してからセカンダリメッセージを指定すること。

【呼出し形式】

```
void Opmsgcode(int ictg, int msgid)
```

【入力引数】

ictg	メッセージタイプ * 100 + コマンドカテゴリ メッセージタイプ 0 または 1 : プライマリメッセージ プロンプト領域の１行目に表示する 2 : セカンダリメッセージ プロンプト領域の２行目に表示する コマンドカテゴリ (1, 3, 4, 5)
msgid	メッセージ番号

7.3.6 指定ゾーン情報を抽出・設定

指定ゾーンの情報を抽出、または設定する。

【呼出し形式】

```
int Menuzone(int iswt, int iz, MENUZONE* zone)
```

【入力引数】

iswt　　抽出か設定かのスイッチ
　　1　　：設定する，2 ： 抽出する
iz　　ゾーン番号
　　1　　：グラフィックゾーン
　　2　　:MSC ゾーン
　　6　　：メッセージゾーン
　　7　　：プロンプトゾーン

【入出力引数】

zone　　指定ゾーンの情報
　　iswt == 1 のときは入力引数
　　iswt == 2 のときは出力引数
　　zone.xmin　ゾーンの左下座標 X （ラスター）
　　zone.ymin　ゾーンの左下座標 Y （ラスター）
　　zone.xmax　ゾーンの右上座標 X （ラスター）
　　zone.ymax　ゾーンの右上座標 Y （ラスター）
　　zone.csize　文字高さ（ラスター）
　　zone.hrow　１行の高さ（ラスター）
　　zone.wcol　１列の幅（ラスター）
　　zone.nrow]　行数
　　zone.ncol　列数
　　zone.status　ゾーンが動作中かどうか
　　1 ： 動作中、-1 ： 動作していない
　　ゾーン番号が 32 以下のときは必ず１となる。
　　ゾーン番号が 33 以上（マクロ テンポラリウインドウ）のときは，1 または -1 となる。

【戻り値】

0　　：正常
1　　：ゾーン番号が範囲外
2　　：指定ゾーンは定義されていない

7.4　テンポラリポイント

与えられた点と現在のテンポラリポイント作成メソッドでテンポラリポイントを作成する。
テンポラリポイント作成メソッドが交点や投影点などで１点ではテンポラリポイントが作成できないときは「テンポラリポイント作成コマンド（割り込みコマンド)」を起動して呼び出し元に戻る。

テンポラリポイント作成メソッドについては第 8 章の AwGeomParam クラスの説明を参照してください。

【呼出し形式】

```
int PickPoint(TOKEN* token, G2Point* pans)
int IdentPoint(TOKEN* token, DPOINT* pans)
```

【入力引数】

token　　トークン
　　token.typ
　　TknCOD　　：　指定された点をテンポラリポイントにする。

TknDIG : テンポラリポイントモードが自動点や端点などで１点でテンポラリポイントが作成できるときはテンポラリポイントを作成する。テンポラリポイントモードが交点や投影点などで１点ではテンポラリポイントが作成できないときは「テンポラリポイント作成コマンド（割り込みコマンド)」を起動する。
TknPNT : テンポラリポイント作成コマンドからの完了通知。

token.pnt
token.typ が TknCOD,TknDIG のとき: 座標値。

token.scl
token.typ が TknPNT のとき : テンポラリポイント作成コマンドからの完了通知。
IDENT_SUCCESS : 正常。
IDENT_FAILURE : 作成できなかった。

【出力引数】

pans　テンポラリポイント。

【戻り値】

token.typ が TknCOD のとき
IDENT_SUCCESS : テンポラリポイントが作成できた。
IDENT_FAILURE : 引数が不正。
token.typ が TknDIG のとき
IDENT_SUCCESS : テンポラリポイントが作成できた。
IDENT_FAILURE : テンポラリポイントが作成できない または 引数が不正。
IDENT_CONTINUE : テンポラリポイント作成コマンドが起動された。
token.typ が TknPNT のとき
IDENT_SUCCESS : テンポラリポイントが作成できた。
IDENT_FAILURE : テンポラリポイントが作成できない または 引数が不正。

例.「2.14 テンポラリポイントの作成」を参照。

7.5 アイテムピック

● 関数一覧

関数名	機能
IdentItem	アイテムをピックする。
IdentItemCandidate	IdentItem() 関数の次候補アイテムを調べる。
IdentInfoCount	ピックされたアイテムの詳細情報の数を得る。
IdentInfo	ピックされたアイテムの詳細情報を得る。
IdentItems	複数アイテムの自動選択。
IdentItemsCandidate	IdentItems() 関数の次候補アイテムを調べる。
identdb	ピック領域のアイテムをピックする。
identsetbox	矩形のピック領域を設定する。
identsetply	多角形のピック領域を設定する。
IdentCount	ピックされたアイテム数を得る。
IdentIdptrs	ピックされたアイテムのアイテム識別子を得る。

7.5.1 アイテムのピック

アイテムをピックする。

【呼出し形式】

```
const AwItem* PickItem(int cmdctg, const TOKEN& token, int iswt)
int IdentItem(int cmdctg, const TOKEN* token = NULL, int iswt = 0)
```

【入力引数】

cmdctg　コマンドカテゴリ。(1, 3, 4, 5)
token　トークン
　token.typ
　　TknCMD または token が NULL:
　　　初期化する。
　　　・次候補アイテム情報のクリア
　　TknCOD、TknDIG　：　指定された座標でアイテムを選択する。
　　TknSPC　：　次候補アイテムを選択する。
　　TknIDP、TknSCL　：　指定されたアイテム識別子でアイテムを選択する。
　token.pnt
　　token.typ が TknCOD、TknDIG のとき ： 座標値。
　token.scl
　　token.typ が TknIDP、TknSCL のとき ： アイテム識別子。
iswt　アテンションボックスを表示するかどうか。
　　0　：表示しない。
　　1　：表示する。

【戻り値】

IdentItem() 関数
　token.typ が TknCOD、TknDIG、TknSPC、TknIDP、TknSCL のとき、選択されたアイテムのアイテム識別子。アイテムが選択できないか引数が不正の時は 0。
　token.typ が TknCMD のときは常に 0。
PickItem() 関数
　パーマネントアイテムへのポインタを返します。アイテムが選択できないときなどは null を返します。

【補足】

次候補アイテムの有無を調べるには IdentItemCandidate() の戻り値をみる。値が 2 以上ならば次候補アイテムが存在する。詳細な情報が必要なときは IdentInfo() 関数を参照。

例

「2.14 アイテムの選択」を参照。

7.5.2 IdentItem() 関数の次候補を調べる

IdentItem() 関数でピックされたアイテムに、次候補アイテムが存在するかどうかを調べる。

【呼出し形式】

```
int IdentItemCandidate(int cmdctg)
```

【入力引数】

cmdctg　コマンドカテゴリ。(1, 3, 4, 5)

【戻り値】

次候補アイテムの有無。(0, 1-n)
0　：アイテムがピックできなかった。次候補アイテムは存在しない。
1　：アイテムがピックできた。近傍に、次候補アイテムは存在しない。
2-　：アイテムがピックできた。近傍に、次候補アイテムが存在する。

7.5.3　ピックされたアイテムの詳細情報の数を得る

IdentItem() 関数でピックされたアイテムの詳細情報の数を得る。

【呼出し形式】

```
int IdentInfoCount(void)
```

【戻り値】

IdentItem() 関数でピックされたアイテムの詳細情報の数。(0, 1-n)
0　：詳細情報がない。（アイテムがピックできなかった）
1　：詳細情報は一つだけ存在する。（アイテムが一つだけピックできた）
2-　：詳細情報は複数存在する。（近傍にアイテムが複数あった）
これはピックされた位置とアイテムとの距離の近い順に並んでおり
IdentItem、IdentItems の次候補要求で順次選択される。

【補足】

一度のピックで１アイテムだけを選択する場合はこの関数で数を調べる必要はなく、詳細情報の最初の一つだけを参照すればよい。
たとえば一点トリムコマンドのように一度のピックで二つのアイテムが必要な場合にこの関数で数を調べればよい。

7.5.4　ピックされたアイテムの詳細情報を得る

IdentItem() 関数でピックされたアイテムの詳細情報配列のポインタを得る。

【呼出し形式】

```
IDENTINFO* IdentInfo(void)
```

【戻り値】

IdentItem() 関数でピックされたアイテムの詳細情報配列のポインタ。
NULL　：詳細情報がない。（IdentItem でアイテムがピックできなかった）

IDENTINFO 構造体のメンバ（IDENTINFO 構造体は acaddef.h 内で定義している）。

int idptr	アイテム識別子 (1 ー)。
int snum	ピックしたサブレコードのインデックス (0 ー)。
int styp	ピックしたサブレコードの種類。
int wend	幾何図形要素のどちらの端点に近い方を指示したか。 1 = 始点、-1 = 終点。ピックしたサブレコードの種類が線分・円・Bezier セグメントのときのみ有効。 その他のときは 1。
double dist	指示した位置と、カーブ上に投影した点との距離。
DPOINT ploc	指示した位置を、ピックしたカーブ上に投影した点。

DGEOM geom　　サブレコードの内容。

サブレコードの種類	データ
AwSr::POINT	P
AwSr::LINE	Ps, Pe
AwSr::CIRCLE	Ps, Pm, Pe, Pc, radius, angle
AwSr::BEZIER	Q1, Q2, Q3, Q4, length
AwSr::TEXT	P0, P1, P2, P3, Px Px は文字列位置
AwSr::MARK	P0, P1, P2, P3, Px Px はスナップノード位置

int ctgsridx　　ピックしたサブレコードの前に現れた分類サブレコード
(AwSr::CATEGORY) のインデックス (-1 : なし, 0 ー)。
short pid　　図面配置状態のとき、ピックされたアイテムの属するピクチャ番号。(1 - 512)
0 = 図面配置状態ではない。
short wid　　図面配置状態のとき、ピックされたアイテムの属するウインドウ番号。

例
```
const IDENTINFO* info = IdentInfo();
int idptr1 = info->idptr;    /* 一番近いアイテムのアイテム識別子 */
idptr1 = info[0].idptr;      /* 上の行と同じ。どちらでもよい。*/
int idptr2 = info[1].idptr;  /* 二番目に近いアイテムのアイテム識別子 */
```

7.5.5　複数アイテムの自動選択

まず、与えられた点でアイテムをピックする。
アイテムがピックできないときは矩形または多角形領域でのアイテムの選択であるものとし「複数アイテムの自動選択コマンド (割り込みコマンド)」を起動して呼び出し元に戻る。
選択されたアイテムはハイライトアイテムリストに追加される。

【呼出し形式】

```
int IdentItems(int cmdctg, TOKEN *token = NULL, int flag = 0)
```

【入力引数】

cmdctg　　コマンドカテゴリ。(1, 3, 4, 5)
token　　トークン
token.typ
TknCMD または token が NULL:
初期化する。
・次候補アイテム情報のクリア
・最後の操作情報のクリア
TknCOD、TknDIG　　:　指定された座標でアイテムをピックする。
ピックできないときは矩形または多角形での領域指定とみなし「複数アイテムの自動選択コマンド (割り込みコマンド)」を起動する。
選択されたアイテムはハイライトアイテムリストに追加される。
TknDIG のときに token.scl が 1 のとき (コントロール／シフトキーが押されていたとき) は選択されたアイテムをハイライトアイテムリストから排除する。

TknSPC : 次候補アイテムがあれば、前に追加されたアイテムをハイライトアイテムリストから排除し、次候補アイテムをハイライトアイテムリストに追加する。
TknBSP : 前回の操作を元に戻す。さらに続けて TknBSP が渡されるとハイライトアイテムリストに追加されているアイテムを一つづつ排除する。
TknIDP、TknSCL : 指定されたアイテム識別子をハイライトアイテムリストに追加する。
TknITM : 複数アイテムの自動選択コマンドからの完了通知。
TknEOC : 今回処理したアイテムのアイテム識別子を次回以降に参照するために保存しておく。保存対象は３番目の引数 flag で指定する。
TknMDF : TknEOC で保存されていたアイテム識別子（PRV）またはアクティブリスト内のアイテム識別子（USEACT）をハイライトアイテムリストに追加する。
どちら（PRV か USEACT）を追加するかは token.cid で指定する。

token.cid
token.typ が TknMDF のとき : コマンド修飾子のコマンド番号。
PRV [34, 1, 207]
USEACT [34, 1, 1]

token.pnt
token.typ が TknCOD、TknDIG のとき:
座標値。

token.scl
token.typ が TknDIG のとき : コントロールキー／シフトキーが押されていたかどうか。
0 : 押されていない（アイテムの追加）
1 : 押されていた（アイテムの排除）
token.typ が TknIDP、TknSCL のとき:
アイテム識別子。
token.typ が TknITM のとき : 複数アイテムの自動選択コマンドからの完了通知。
IDENT_SUCCESS : 正常。
IDENT_FAILURE : 選択できなかった。

flag token.typ が TknEOC のとき、保存対象を指定する。
SAVE_LST_ITM : ハイライトアイテムリストの内容。
SAVE_NEW_ITM : 新たに作成されたアイテムのアイテム識別子。
SAVE_ACT_ITM : アクティブリスト内のアイテム識別子。
たとえば移動コマンド（MOVE）で「複製しない」ときは SAVE_LST_ITM または SAVE_ACT_ITM とし、「複製する」ときは SAVE_NEW_ITM としている。

【戻り値】

token.typ が TknCOD、TknDIG のとき
IDENT_SUCCESS : アイテムが選択された。
IDENT_FAILURE : 引数が不正。
IDENT_CONTINUE : 複数アイテムの自動選択コマンドが起動された。
token.typ が TknSPC、TknIDP、TknSCL、TknBSP、TknMDF、TknITM のとき
IDENT_SUCCESS : アイテムが選択された。
IDENT_FAILURE : アイテムが選択できない または 引数が不正。
token.typ が TknEOC のとき
IDENT_SUCCESS : アイテム識別子が保存された。
IDENT_FAILURE : アイテム識別子が保存できない または 引数が不正。
token.typ が TknCMD のとき
IDENT_SUCCESS : 正常。
IDENT_FAILURE : 引数が不正。

【補足】

次候補アイテムの有無を調べるには IdentItemsCandidate() の戻り値をみる。値が 2 以上ならば次候補アイテムが存在する。

例

「2.14 複数アイテムの自動選択」を参照。

7.5.6 IdentItems() 関数の次候補を調べる

IdentItems () 関数で、与えられた点でアイテムがピックできた場合、ピックされたアイテムの次候補アイテムが存在するかどうかを調べる。矩形および多角形領域での選択の場合は戻り値が 0 になる。

【呼出し形式】

```
int IdentItemsCandidate(int cmdctg)
```

【入力引数】

cmdctg　　コマンドカテゴリ。(1, 3, 4, 5)

【戻り値】

次候補アイテムの有無。(0, 1-n)

0　　：与えられた点でアイテムがピックできなかった。または、矩形か多角形領域での選択。
1　　：アイテムがピックできた。近傍に、次候補アイテムは存在しない。
2-　　：アイテムがピックできた。近傍に、次候補アイテムが存在する。

7.5.7 ピック領域のアイテムをピックする

ピック領域内のアイテムをピックする。予めピック領域を identsetbox() または identsetply() 関数で指定しておく。選択マスク、表示マスクを参照し選択可能、表示可能なアイテムのみを対象とする。ピックしたアイテムの識別子は IdentIdptrs() 関数を参照のこと。

【呼出し形式】

```
int identdb(int type, int flag, int vpid)
```

【入力引数】

type　　ピックの方法
　　2　　:identsetbox() で設定した矩形内。
　　3　　:identsetply() で設定した多角形内。
flag　　3 とする。
vpid　　ビューポート番号 (1-N)。0 はアクティブビューポート。

【戻り値】

ピックしたアイテム数 (0, 1-N)

0　　：アイテムがピックできない

7.5.8 矩形のピック領域設定

矩形のピック領域を設定する。この関数で設定した領域は identdb() 関数で使用される。

【呼出し形式】

```
void identsetbox(int flag, const DPOINT* p1, const DPOINT* p2)
```

【入力引数】

flag　　フラッグ。0 とする。

p1, p2　　矩形の対角点

7.5.9　多角形のピック領域設定

多角形のピック領域を設定する。この関数で設定した領域は identdb () 関数で使用される。

【呼出し形式】

```
void identsetply(int npnt, const DPOINT pbuf[])
```

【入力引数】

npnt　　多角形の点数 (3― 128)
pbuf　　多角形の点列 (始点と終点は重複させない)

7.5.10　ピックされたアイテムの個数を得る

identdb() 関数でピックされたアイテムの個数を得る。

【呼出し形式】

```
int IdentCount(void)
```

【戻り値】

identdb () 関数でピックされたアイテムの個数（identdb() の戻り値と同じ)。(0, 1-N)
0　　：アイテムがピックできなかった

7.5.11　ピックされたアイテムのアイテム識別子を得る

identdb() 関数でピックされたアイテムのアイテム識別子のポインタを得る。

【呼出し形式】

```
const int* IdentIdptrs(void)
```

【戻り値】

ピックされたアイテムのアイテム識別子配列のポインタ。
　null ポインタ　　：　アイテムがピックできなかった。

例.

```
DRECT rect = { { 0.0, 0.0 }, { 200.0, 100.0 } };
identsetbox(0, &rect.pmin, &rect.pmax);
int nitem = identdb(5, 3, 0);
if (nitem > 0) {
  const int* idptrs = IdentIdptrs();
  int idptrFirst = idptrs[0];
  int idptrLast  = idptrs[nitem - 1];
}
```

7.6　表示色

● 関数一覧

CIr004	**カラーテーブルの値を設定する。**
CIr104	**カラーテーブルの値を得る。**

7.6.1　カラーテーブルの設定

カラーテーブルの値を設定する。

【呼出し形式】

```
int Clr004(int iclr, const short igrb[], int iswt)
```

【入力引数】

```
iclr      カラーの種類
           -4              :  バックグラウンド（ドローイングレイアウト）カラー
           -3              :  バックグラウンドカラー
           -2              :  グリッドのカラー
           -1              :  テンポラリ図形のカラー
           0               :  ラバーバンドカラー
           1 - MAXITMCLR   :  実図形のカラー
igrb      カラーテーブルの値
           igrb[0]         :  緑の輝度 (0 - 255)
           igrb[1]         :  赤の輝度 (0 - 255)
           igrb[2]         :  青の輝度 (0 - 255)
iswt      0。
```

【戻り値】

```
0       : 正常
1       : エラー（引数の値が不正）
```

例

```
/* 実図形のカラー#１を赤にする */
short igrb[3] = { 0, 255, 0};
Clr004(1, igrb, 0);
```

7.6.2　カラーテーブル値の取得

カラーテーブルの値を得る。

【呼出し形式】

```
int Clr104(int iclr, short igrb[])
```

【入力引数】

```
iclr      カラーの種類
           -4              :バックグラウンド（ドローイングレイアウト）カラー
           -3              :バックグラウンドカラー
           -2              :グリッドのカラー
           -1              :テンポラリ図形のカラー
           0               :ラバーバンドカラー
           1 - MAXITMCLR   :実図形のカラー
```

【出力引数】

```
igrb      カラーテーブルの値
           igrb[0]         :  緑の輝度 (0 - 255)
           igrb[1]         :  赤の輝度 (0 - 255)
           igrb[2]         :  青の輝度 (0 - 255)
```

【戻り値】

0　　：正常
1　　：エラー（引数の値が不正）

例

```
/* 実図形カラー #1 のカラーテーブルの値を得る */
short igrb[3];
Clr104(1, igrb);
```

7.7　スクリーンレイアウト

● 関数一覧

Slo001	**アクティブスクリーンレイアウトを切り換える。**
Slo101	**アクティブスクリーンレイアウトの番号を得る。**
Vie001	**アクティブビューポートを切り換える。**
Vie101	**アクティブビューポートの番号を得る。**
Viechang	**アクティブビューポートを一時的に切り換える。**
Pic001	**アクティブピクチャを切り換える。**
Pic102	**指定ビューポートのピクチャ番号を得る。**
Lst002	**アイテム属性ごとのアイテム数を得る**
Lst003	**スクリーンレイアウト情報を得る**

7.7.1　アクティブスクリーンレイアウトの切り換え

アクティブスクリーンレイアウトを切り換える。

【呼出し形式】

```
int Slo001(int islo)
```

【入力引数】

islo　スクリーンレイアウトの番号 (1 - 31)

【戻り値】

0　　：正常
1　　：エラー（引数の値が不正）

例 .

```
/* スクリーンレイアウトを 5 にする */
Slo001(5) ;
```

7.7.2　アクティブスクリーンレイアウト番号の取得

アクティブスクリーンレイアウト番号を得る。

【呼出し形式】

```
int Slo101(void)
```

【戻り値】

アクティブスクリーンレイアウト番号 (1 - 31)

7.7.3　アクティブビューポートの切り換え

アクティブビューポートを切り換える。

【呼出し形式】

```
int Vie001(int ivie)
```

【入力引数】

ivie	アクティブにするビューポートの番号 (1 - 128)

【戻り値】

0	: 正常
1	: エラー（引数の値が不正）

例.

```
/* アクティブビューポートをビューポート５にする */
Vie001(5);
```

7.7.4　アクティブビューポート番号の取得

アクティブビューポートの番号を得る。

【呼出し形式】

```
int Vie101(void)
```

【戻り値】

アクティブビューポートの番号 (1 - 128)

7.7.5　アクティブビューポートの一時切り換え

アクティブビューポートを一時的に切り換える。アクティブビューポート以外がピックされたときに、ピックされたビューポートを一時的にアクティブビューポートにする場合に使用する。処理が終了したらアクティブビューポートを元に戻しておくこと。

【呼出し形式】

```
void Viechang(int ivie, int ista)
```

【入力引数】

ivie	アクティブにするビューポートの番号 (1 - 128)
ista	必ず 0 をセットすること。

例.

```
if (token->typ == TknDIG) { /* ディジタイズ座標 */
  /* アクティブビューポート番号を得る */
  int actvie = Vie101();
  /* ディジタイズされたビューポート番号を得る */
  int tknvie = token->vp;
^ /* ピックされたビューポートがアクティブビューポートではないとき、
   *アクティブビューポートを一時的にピックされたビューポートにする */
  if (tknvie != actvie)
    Viechang(tknvie, 0);
```

```
    /* ここで必要な処理を行う */

    /* アクティブビューポートを元に戻す */
    if (tknvie != actvie)
      Viechang(actvie);
}
```

7.7.6　アクティブピクチャの切り換え

アクティブピクチャを切り換える。

【呼出し形式】

```
int Pic001(int ipic)
```

【入力引数】

ipic　アクティブにするピクチャの番号 (1 - AwItemAttributes::MAX_PICTURE)

【戻り値】

0　　：正常
1　　：エラー（引数の値が不正）

例

```
    /* アクティブピクチャをピクチャ５にする */
    Pic001(5);
```

7.7.7　指定ビューポートのピクチャ番号を取得

指定したビューポートが表示するピクチャ番号を得る。

【呼出し形式】

```
int Pic102(int ivie)
```

【入力引数】

ivie　　ビューポート番号。(1-N)

【戻り値】

ピクチャ番号。(0、1-N)
0　　：　指定されたビューポートが存在しない。

7.7.8　アイテム属性ごとのアイテム数

アイテム属性 (タイプ、クラス、レビジョン、線種、線幅) の使用状況を得る。

【呼出し形式】

```
int Lst002(int iflg, int ipic, int itms[])
```

【入力引数】

iflg　　使用状況を得る属性の種類
1　　：アイテム種別（点、線、円弧...）の使用状況を得る
2　　：クラスの使用状況を得る
3　　：レビジョンの使用状況を得る

```
        4       :線種の使用状況を得る
        5       : 線幅の使用状況を得る
ipic    使用状況を調べるピクチャのピクチャ番号
        -1              :  全ピクチャ
        0               :  アクティブピクチャ
        1 - AwItemAttributes::MAX_PICTURE:ピクチャ番号
```

【出力引数】

```
itms    アイテム数の並び
        int itms[N]
        iflg == 1 の時、N は MAXITMTYP
        iflg == 2 の時、N は MAXITMCLS
        iflg == 3 の時、N は MAXITMREV
        iflg == 4 の時、N は MAXITMLFT
        iflg == 5 の時、N は MAXITMLWT
```

【戻り値】

0　　: 正常

例

```
/* 全ピクチャのクラスの使用状況を得る */
int itms[MAXITMCLS];
Lst002(2, -1, itms);

/* クラス１のアイテム数が５、クラス３のアイテム数が７で、
   他のクラスにアイテムが存在しない時は次の値が戻る。
itms[0] : 5
itms[1] : 0
itms[2] : 7
itms[3] ～ itms[MAXITMCLS-1] : 0
*/
```

7.7.9　スクリーンレイアウト情報の取得

スクリーンレイアウト情報を得る。

【呼出し形式】

```
int Lst003(int islo, int tbl[][3], double zon[][4])
```

【入力引数】

```
islo    情報を抽出するスクリーンレイアウト番号
        -1              :  全スクリーンレイアウト
        0               :  アクティブスクリーンレイアウト
        1 - 31          :  スクリーンレイアウト番号
```

【出力引数】

```
tbl     スクリーンレイアウト情報 (int tbl[128][3])
        スクリーンレイアウト番号、ビューポート番号の小さい順に並べられる。
        tbl[][0] : スクリーンレイアウト番号
        tbl[][1] : ビューポート番号
        tbl[][2] : ピクチャ番号
zon     表示範囲テーブル (double zon[128][4])
        zon[][0] : 表示範囲 最小 X 座標 ( モデル座標 )
        zon[][1] : 表示範囲 最小 Y 座標 ( モデル座標 )
```

zon[][2] : 表示範囲 最大 X 座標（モデル座標）
zon[][3] : 表示範囲 最大 Y 座標（モデル座標）

【戻り値】

ビューポート数 (1 - 128)。エラーの場合は 0。

例

```
/* アクティブスクリーンレイアウトの状態を得る */
int tbl[128][3];
double zon[128][4];
Lst003(0, tbl, zon);
```

7.8 画面表示

● 関数一覧

Vieerase	**指定ビューポートの指定プレーンを消去する。**
rpttmppln	**テンポラリ・プレーンを再表示する。**
Rpt101	**画面の再表示を行う。**
Rptitems	**指定されたアイテムを画面上に表示または消去する。**
Pan101	**画面表示部分の移動を行う。**
Zom101	**画面表示のズーミングを行なう。**

7.8.1 画面表示部分の移動

画面表示部分の移動を行なう。

【呼出し形式】

```
int Pan101(int form, const DPOINT* pnt)
```

【入力引数】

form　移動の分類
　1　：移動量を指定 (PAN P1-P2)
　2　：表示中心位置を指定 (PAN/CTR)

pnt　移動量または表示中心位置
　form == 1 のときの移動量（モデル座標系）
　　pnt->x　：　X 方向の移動量
　　図形を向かって右側に移動（カメラを左側に移動）するとき正の値、反対方向のとき負の値とする。
　　pnt->y　：　Y 方向の移動量
　　図形を上側に移動（カメラを下側に移動）するとき正の値、反対方向のとき負の値とする。
　form == 2 のとき表示中心位置（モデル座標）
　　pnt->x　：　表示中心位置 X
　　pnt->y　：　表示中心位置 Y

【戻り値】

0　：正常
1　：エラー（引数の値が不正）

7.8.2 画面の再表示

画面の再表示を行う。

【呼出し形式】

```
void Rpt101(int iopt)
```

【入力引数】

iopt	なにを再表示するかを指定する。	
	1	：アクティブピクチャの再表示。
	2	：全てのビューポートの再表示。
	3	：メニューを含めた画面全体の再表示。
	4	：アクティブビューポートの再表示。

7.8.3 アイテムの表示・消去

アイテムを画面上に表示 / 消去する。

【呼出し形式】

```
void Rptitems(int key, int mode, int *idptrs, int nidptr)
```

【入力引数】

key	スクリーンの選択及び表示データの選択	
	0	：テンポラリ図形プレーンに表示
	1	：実図形プレーンに表示
	-1	：テンポラリ図形プレーンにアイテムの構成点を表示
mode	表示か消去かの選択	
	0	：表示
	1	：消去
idptrs	アイテム識別子の並び	
nidptr	アイテム識別子の数	

例 . テンポラリ図形プレーンにアイテム識別子 101 と 102 のアイテムを表示する。

```
const int idptrs[] = { 101, 102};
Rptitems(0, 0, idptrs, 2);
```

7.8.4 指定ビューポートの指定プレーン消去

指定ビューポートの、指定プレーンを消去する。

【呼出し形式】

```
void Vieerase(int ivie, int plane)
```

【入力引数】

ivie	ビューポート番号	
	0	：全部のビューポート
plane	プレーン指定	
	GRIDSCREEN	： グリッドプレーン
	ACTSCREEN	： 実図形プレーン
	TEMPSCREEN	： テンポラリ図形プレーン
	RUBBSCREEN	： ラバーバンドプレーン

例 . 全てのビューポートのテンポラリ図形プレーンを消去する。

```
Vieerase(0, TEMPSCREEN);
```

7.8.5　テンポラリ・プレーンを再表示

テンポラリ・プレーンを再表示する。

【呼出し形式】

```
void rpttmppln()
```

7.8.6　表示画面のズーミング

画面表示のズーミングを行なう。

【呼出し形式】

```
int Zom101(int form, const double* dprm)
```

【入力引数】

form　　ズーム操作の種類
- 1　　:ZOOM/ALL
- 2　　:ZOOM P1-P2
- 3　　:ZOOM/UP
- 4　　:ZOOM/DOWN
- 5　　:ZOOM/BACK
- 11　　:ZOOM/ALLVIE

dprm　　ズームのパラメータ。下記以外のときは参照しない。

ZOOM P1-P2 のとき
- dprm[0]　：画面に表示する領域 左下 x
- dprm[1]　：画面に表示する領域 左下 y
- dprm[2]　：画面に表示する領域 右上 x
- dprm[3]　：画面に表示する領域 右上 y

ZOOM SCALE のとき
- dprm[0]　：今の表示状態に対する拡大／縮小率。たとえば倍にするときは 2.0 とする。

【出力引数】

0　：正常
1　：エラー（引数の値が不正）

7.9　縮尺

7.9.1　ピクチャ縮尺値・ドローイング縮尺値を設定

ピクチャ縮尺値またはドローイング縮尺値を設定する。

【呼出し形式】

```
void Scfval001(int ipic, double scf)
```

【入力引数】

ipic　　ピクチャ縮尺値またはドローイング縮尺値の選択
- -1　　:ドローイング縮尺値
- 0　　:アクティブピクチャのピクチャ縮尺値
- 1 - AwItemAttributes::MAX_PICTURE:指定ピクチャのピクチャ縮尺値

scf　　縮尺値

例 . ドローイング縮尺値を 1/2 に設定する。

```
Scfval001(-1, 0.5);
```

7.9.2 ピクチャ縮尺値・ドローイング縮尺値の取得

ピクチャ縮尺値またはドローイング縮尺値を取り出す。

【呼出し形式】

```
double Scfval101(int ipic)
```

【入力引数】

ipic　　ピクチャ縮尺値またはドローイング縮尺値の選択
　　-1　　：ドローイング縮尺値
　　0　　：アクティブピクチャのピクチャ縮尺値
　　1 - AwItemAttributes::MAX_PICTURE:指定ピクチャのピクチャ縮尺値

【戻り値】

縮尺値

例.アクティブピクチャのピクチャ縮尺値を取り出す

```
double picscf = Scfval101(0);
```

7.10 ラバーバンドとドラッグ

● 機能一覧

関数名	機能
Rubset	ラバーバンドを設定または解除する。
Dragopen	ドラッギングを初期化する。
Dragend	ドラッギングを終了させる。

ラバーバンドとドラッギングは同時には使用できません。どちらか一方だけです。

7.10.1 ラバーバンドの設定または解除

線分、円／円弧、矩形、長さ寸法のラバーバンドを設定または解除する。

【呼出し形式】

```
void Rubset(int ictg, int type, int iswt, const double dprm[])
```

【入力引数】

ictg　　コマンドカテゴリ。(1, 3, 4, 5)
type　　ラバーバンドのタイプ番号
iswt　　ラバーバンドの設定／解除
　　0　　：ラバーバンドを解除する。
　　1　　：ラバーバンドを設定する。
　　2　　：ラバーバンドデータを追加する。
dprm　　ラバーバンドデータ
　　iswt == 1 または iswt == 2 のときだけ必要。

ラバーバンドタイプとデータの設定方法

type 1　線分ラバーバンド
iswt == 1 で、線分の始点座標（dprm）を設定すると、ラバーバンドを開始する。
type 2　矩形ラバーバンド
iswt == 1 で、矩形のコーナー座標（dprm）を設定すると、ラバーバンドを開始する。
type 3　円ラバーバンド（中心－円周点）
iswt == 1 で、円の中心点座標（dprm）を設定すると、ラバーバンドを開始する。
type 4　円弧ラバーバンド（始点－通過点－終点）
iswt == 1 で、円の始点座標（dprm）を設定すると、線分ラバーバンドを開始する。
iswt == 2 で、円の通過点座標（dprm）を設定すると、３点円弧ラバーバンドになる。
type 5　長さ寸法
iswt == 1 で、サブタイプと寸法両端点、文字枠の矩形（dprm）を設定すると、寸法線のラバーバンドを開始する。文字枠をつけたくないときは矩形の 4 点を寸法端点（1 点目）と同じ座標にセットする。
サブタイプ（1 : 水平寸法、2 : 垂直寸法、3 : 平行寸法）
type 7　円弧ラバーバンド（始点－終点－通過点）
iswt == 1 で、円の始点座標（dprm）を設定すると、線分ラバーバンドを開始する。
iswt == 2 で、円の終点座標（dprm）を設定すると、始終点を固定した３点円弧ラバーバンドになる。
type 8　線分ラバーバンド
X 軸または Y 軸に平行な線分のラバーバンドをする。線分が X 軸と成す角度が、Y 軸との角度よりも小さければ、X 軸に平行な線分のラバーバンドとなる。
補助座標系が有効なときは、補助座標系の X 軸、Y 軸を使用する。設定方法は type 1 と同じ。
type 9　矩形ラバーバンド（中心点－対角点）
iswt == 1 で、矩形の中心座標（dprm）を設定すると、矩形の辺が補助座標形の X 軸および Y 軸に平行になるようにラバーバンドを開始する。
type 12　矩形ラバーバンド（対角点－対角点）
矩形の辺が補助座標系の X 軸と Y 軸に平行になるようにラバーバンドする。設定方法は type 2 と同じ。
type 13　円弧ラバーバンド（中心－始点－終点）
iswt == 1 で、円の中心点座標と半径（dprm）を設定すると、円弧の始点がラバーバンドになる。
iswt == 2 で、円の始点座標（dprm）を設定すると、円弧の終点がラバーバンドになる。
type 14　円ドラッギング
設定するデータはない。iswt == 1 でラバーバンドタイプだけを設定する。現在の半径の値が使用される。

```
/* 線分ラバーバンド開始 */
const double ps[] = {100.0, 50.0};
Rubset(1, 1, 1, ps);
return;

/* ラバーバンド解除 */
Rubset(1, 1, 0, (const double *)NULL);
return;

/* (100,10) - (200,30) を両端点とした垂直寸法のラバーバンド */
/* 文字枠はつけない */
const double dprm[] = { 2, 100.0, 10.0, 200.0, 30.0,
                        100.0, 10.0, 100.0, 10.0, 100.0, 10.0, 100.0, 10.0};
Rubset(1, 5, 1, dprm);
return;
```

7.10.2　ドラッギングを初期化

ドラッギングを初期化する。

【呼出し形式】

```
AwDragLoader* Dragopen(int ictg, int mode, const DPOINT pnts[])
```

【入力引数】

ictg　　コマンドカテゴリ。(1, 3, 4, 5)
mode　　ドラッギングのモード
　　1　　：移動ドラッギング
　　2　　：回転ドラッギング
　　3　　：水平移動ドラッギング
　　4　　：垂直移動ドラッギング
pnts　　ドラッギング基点 (DPOINT pnts[2])
　　pnts[0]　　：　移動中心点座標
　　pnts[1]　　：　回転中心点座標
　　移動ドラッギングのときは、回転中心点座標＝移動中心点座標とすること。

【戻り値】

AwDragLoader オブジェクトへのポインタを返します。引数に誤りがあると null ポインタを返します。この時点ではドラッギングするデータは空です。このオブジェクトを使ってドラッギングするデータを登録します。

7.10.3 ドラッギングの終了

ドラッギング（あるいはラバーバンド）を終了させる。

【呼出し形式】

```
void Dragend(int ictg)
```

【入力引数】

ictg　　コマンドカテゴリ。(1, 3, 4, 5)

7.11 AwDragLoader クラス

このクラスはドラッグする図形を登録するクラスです。

7.11.1 アイテム

アイテムをドラッギングデータに追加します。

```
bool LoadItem(const AwItem& item);
```

item　AwItem オブジェクト。

イテレータが与える全てのアイテムをドラッギングデータに追加します。

```
void LoadItems(AwItemListIterator* pIterator);
```

pIterator　AwItemListIterator イテレータへのポインタ。

7.11.2 単純図形

曲線（線分、円弧、3 次 Bezier 曲線）をドラッギングデータに追加します。

```
void Load(const G2CurveArray& curves);
```

curves　曲線オブジェクトの配列オブジェクト。

楕円曲線をドラッギングデータに追加します。

```
bool LoadConic(const G2Conic& conic);
```

conic　　楕円曲線オブジェクト。

矩形をドラッギングデータに追加します。
```
void LoadRect(const G2Rect& rect);
```
rect　　矩形オブジェクト。

多角形をドラッギングデータに追加します。
```
void LoadPolyline(const G2Point points[], int count, bool close = false);
```
points　点配列（重複点がないこと）。
count　点数 (2 <= count)。
close　終点と始点を結ぶとき true とします。その場合点数は３以上とします。

マークをドラッギングデータに追加します。
```
void LoadMark(const AwSrMark& srMark, double drafScale);
```
srMark　　マーク・サブレコードオブジェクト。
drafScale　ドラフティングスケール (0 < drafScale)

グラフィックステキストをドラッギングデータに追加します。
```
void LoadText(const AwSrText& srText, double drafScale);
```
srText　　テキスト・サブレコードオブジェクト。
drafScale　ドラフティングスケール (0 < drafScale)

7.12 AwHighlightSet クラス

このクラスはハイライトセットを表現します。
ハイライトセットはコマンドのレベルごとに在ります。コマンドハンドラが最初に呼び出されたときには、ハイライトセットは初期化されています。ハイライトセットはを得るには GetHighlightSet() 関数を使います。

7.12.1 Getter

ハイライト・アイテム・リストを得ます。
```
AwHighlightItems* GetHlItems();
```
戻り値　AwHighlightItems オブジェクトへのポインタを返します。

ハイライト点列リストを得ます。
```
AwHighlightPoints* GetHlPoints();
```
戻り値　AwHighlightPoints オブジェクトへのポインタを返します。

7.12.2 メソッド

全てのハイライトオブジェクトをクリアし、初期値を設定します。
```
void Clear();
```

全てのハイライトオブジェクトのデータを表示します。
```
void Draw() const;
```

7.13 AwHighlightItems クラス

このクラスはハイライトするパーマネントアイテムの配列を表現します。AwHighlightItems オブジェクトは AwHighlightSet オブジェクトから得ます。
このクラスはアイテム識別子を比較するときその符号を無視します。またアイテム識別子の重複を排除します。

7.13.1 条件設定

このオブジェクトが保持するアイテム数の上限を設定します。変更しなければ
AwPermItemDb::MAX_ITEM_COUNT です。
```
void SetMaxCount(int size);
```
size　　アイテム数の上限。

アイテムの表示方法を設定します。
```
void SetDisplay(int show);
```
show　**アイテムの**表示方法。
- SHOW　　：実際の図形を表示する。(デフォルト)
- SHOW_END　：始終点にマークを表示する。図形は表示しない。
- NOSHOW　　：表示しない。

7.13.2 アイテム識別子の取得

ハイライトリストが空か判定します。
```
bool IsEmpty() const;
```
戻り値　このオブジェクトが空のとき true を返します。

ハイライトリストのアイテム識別子の数が上限に達しているか判定します。
```
bool IsFull() const;
```
戻り値　このオブジェクトのアイテム識別子数が上限以上のとき true を返します。

ハイライトリストのアイテム識別子の数を得ます。
```
int  GetCount() const;
```
戻り値　このオブジェクトのアイテム識別子の数を返します。

ハイライトリストが指定したアイテム識別子を含むか調べます。
```
bool Contains(int idptr) const;
```
idptr　　アイテム識別子。
戻り値　このオブジェクトがアイテム識別子を含むとき true を返します。

アイテム識別子を得ます。
```
int  GetAt(int index) const;
```
index　　アイテム識別子の位置を示すインデックス。0 <= index < GetCount()。
戻り値　index の位置のアイテム識別子を返します。インデックスが範囲外のときは 0 を返します。

アイテム識別子の配列を得ます。
```
const AwItemIdArray& GetItemIds() const;
```
戻り値　AwItemIdArray オブジェクトの参照を返します。

7.13.3 アイテム識別子の追加、削除

アイテム識別子をハイライトリストの最後に追加します。
```
bool Add(int idptr);
```
idptr　　アイテム識別子。
戻り値　追加したとき true を返します。

複数のアイテム識別子をハイライトリストの最後に追加します。
```
bool Append(const AwItemIdArray& itemids);
```
itemids　　AwItemIdArray オブジェクトが含むアイテム識別子を最後に追加します。
戻り値　　追加したとき true を返します。何も追加しなかったら false を返します。

ハイライトリストを空にします。

```
bool RemoveAll();
```
戻り値　このオブジェクトが空なら false、空にしたとき true を返します。

ハイライトリストの最後のアイテム識別子を削除します。
```
bool RemoveLast();
```
戻り値　このオブジェクトが空なら false、削除したとき true を返します。

ハイライトリストから指定したアイテム識別子を削除します。
```
bool Remove(int id);
```
id　　アイテム識別子。
戻り値　削除したとき true を返します。

ハイライトリストから指定したアイテム識別子を削除します。
```
bool Remove(int count, const int ids[]);
```
count　配列内のアイテム識別子の数。
ids　　アイテム識別子を格納した配列。
戻り値　削除したとき true を返します。

7.13.4 表示

SetDisplay() メソッドで設定した表示方法に従い、アイテムをテンポラリ・プレーンに表示します。
ハイライトリストの全てのアイテムを描画します。
```
void DrawAll() const;
```

ハイライトリストの最後のアイテムだけを描画します。
```
void DrawLast();
```

7.14 AwHighlightPoints クラス

このクラスはハイライトする点列を表現します。AwHighlightPoints オブジェクトは AwHighlightSet オブジェクトから得ます。

7.14.1 条件設定

このオブジェクトが保持する点数の上限を設定します。変更しなければ 256 点です。
```
void SetMaxCount(int size);
```
size　　点数の上限。

点列の表示方法を設定します。
```
void SetDisplay(int show);
```
show　　点列の表示方法。
- POLYLINE　　：点列を結ぶ折れ線。(デフォルト)
- RECTANGLE　　：点数が 2 の時、矩形と 1 点目 ― 2 点目を結ぶ直線を表示。
  点数が 3 以上の時、点列を結ぶ折れ線を表示。
- RECT_WCS　　：点数が 2 の時、WCS にそった矩形と 1 点目 ― 2 点目を結ぶ折れ線を表示。
  点数が 3 以上の時、点列を結ぶ折れ線を表示。
- NOSHOW　　：表示しない。

点の位置にマーカーを表示します。
```
void SetMarker(int marker);
```
marker　マーカーの番号。
- 1　　四角 (□)
- 15　　プラス (+)
- 16　　アスタリスク (*)

17　　丸 (○)
SetDisplay() メソッドと SetMarker() メソッドは排他的です。SetMarker() メソッドを使うと SetDisplay() メソッドの設定はクリアされます。その逆も同様です。

7.14.2 点の取得

ハイライトリストが空か判定します。
```
bool IsEmpty() const;
```
戻り値　このオブジェクトが空のとき true を返します。

ハイライトリストの点数が上限に達しているか判定します。
```
bool IsFull() const;
```
戻り値　このオブジェクトの点数が上限以上のとき true を返します。

ハイライトリストの点数を得ます。
```
int  GetCount() const;
```
戻り値　このオブジェクトの点数を返します。

点を得ます。
```
const G2Point& GetAt(int index) const;
```
index　　点の位置を示すインデックス。0 <= index < GetCount()。
戻り値　index の位置の G2Point オブジェクトの参照を返します。インデックスが範囲外のときは G2Point::ORIGIN を返します。

点の配列を得ます。
```
const G2PointArray& GetPoints() const;
```
戻り値　G2PointArray オブジェクトの参照を返します。

7.14.3 点の追加、削除

点をハイライトリストの最後に追加します。
```
bool Add(const G2Point& point);
```
point　　追加する点。
戻り値　追加したとき true を返します。

ハイライトリストのインデックスの位置の点を新しい点で置き換えます。
```
void SetAt(int index, const G2Point& point);
```
index　　点の位置を示すインデックス。0 <= index < GetCount()。
point　　新しい点。

ハイライトリストに点を挿入します。インデックスで示された位置とそれに以降にある点を後方に移動し、インデックスの位置に点を追加します。インデックス 0 は点を先頭に挿入します。
```
bool InsertAt(int index, const G2Point& point);
```
index　　点の挿入位置を示すインデックス。0 <= index <= GetCount()。
point　　挿入する点。
戻り値　点を挿入したら true を返します。

ハイライトリストを空にします。
```
bool RemoveAll();
```
戻り値　このオブジェクトが空なら false、空にしたとき true を返します。

ハイライトリストの最後の点を削除します。
```
bool RemoveLast();
```
戻り値　このオブジェクトが空なら false、削除したとき true を返します。

ハイライトリストの指定したインデックスの点を削除します。
void RemoveAt(int index);
　index　削除する点の位置を示すインデックス。0 <= index <= GetCount()。

7.14.4 表示

SetDisplay() メソッドで設定した表示方法に従い、点列をテンポラリ・プレーンに表示します。
ハイライトリストの全ての点を描画します。
void DrawAll() const;

ハイライトリストの最後の点だけを描画します。
void DrawLast();

7.15 ActiveList クラス

このクラスはアクティブリストを表現します。
アクティブリストはひとつだけ在ります。アクティブリストコマンドがアクティブリストを操作するのが原則です。「ACT/CHN」コマンドは負のアイテム識別子を扱います。モデルファイル読み込みやアクティブモデルのファイルへの保存などでアクティブモデルが変わったときに、アクティブリストはクリアされます。アクティブリストを得るには後述の GetActiveList() 関数を使います。
このクラスはアイテム識別子を比較するときその符号を無視します。アイテム識別子の重複検査は行いませんので、重複するアイテム識別子を追加しないよう注意してください。

7.15.1 アイテム識別子の取得

アクティブリストが空か判定します。
bool IsEmpty() const;
　戻り値　このオブジェクトが空のとき true を返します。

アクティブリストのアイテム識別子の数を得ます。
int GetCount() const;
　戻り値　このオブジェクトのアイテム識別子の数を返します。

アイテム識別子を得ます。
int GetAt(int index) const;
　index　配列のインデックス。0 <= index < GetCount()。
　戻り値　index の位置のアイテム識別子を返します。インデックスが範囲外のときは 0 を返します。

アイテム識別子の配列を得ます。
const AwItemIdArray& GetItemIds() const;
　戻り値　AwItemIdArray オブジェクトの参照を返します。

7.15.2 アイテム識別子の追加

アクティブリストに重複するアイテム識別子を追加しないようにしなければなりません。
アイテム識別子をアクティブリストの最後に追加します。
bool Add(int id);
　id　アイテム識別子。
　戻り値　追加したとき true を返します。

複数のアイテム識別子をアクティブリストの最後に追加します。
bool Append(const AwItemIdArray& ids);
　ids　AwItemIdArray オブジェクト。

戻り値　追加したとき true を返します。何も追加しなかったら false を返します。何も追加しなかったら false を返します。

複数のアイテム識別子をアクティブリストの最後に追加します。
```
bool Append(int count, const int ids[]);
```
count　配列内のアイテム識別子の数。
ids　アイテム識別子を格納した配列。
戻り値　追加したとき true を返します。何も追加しなかったら false を返します。何も追加しなかったら false を返します。

7.15.3 クリア

アクティブリストを空にします。現在のアイテム識別子の配列を保存してからクリアします。
```
void Clear();
```

7.15.4 探索

アクティブリストが指定したアイテム識別子を含むか調べます。アイテム識別子の比較では符号を無視します。
```
int  Find(int id) const;
```
id　アイテム識別子。
戻り値　このオブジェクトが指定のアイテム識別子を含むとき、配列のインデックスを返します。含まないときは -1 を返します。

7.16 オブジェクト取得

7.16.1 GetHighlightSet() 関数

ハイライトセットを得ます。
```
AwHighlightSet* GetHighlightSet(int cmdctg);
```
cmdctg　コマンドカテゴリ。(1, 3, 4, 5)
戻り値　AwHighlightSet オブジェクトへのポインタを返します。引数が誤りなら null を返します。

7.16.2 GetActiveList() 関数

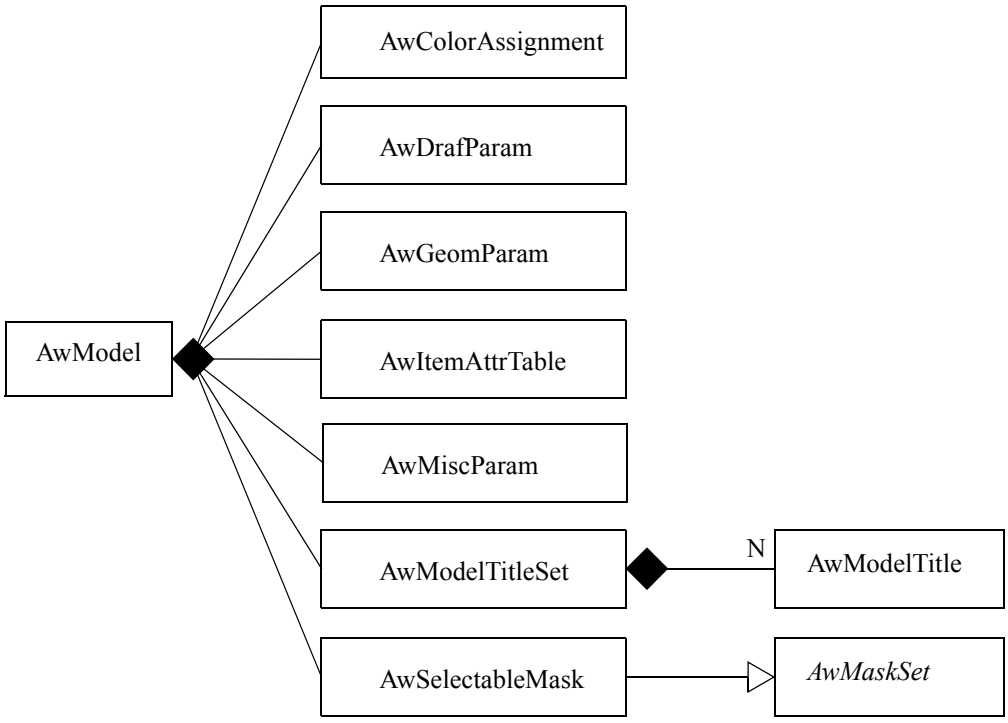
アクティブリストを得ます。
```
AwActiveList* GetActiveList();
```
戻り値　AwActiveList オブジェクトへのポインタを返します。

第 8 章　モデルのパラメータ

この章ではアプリケーションモデル AwModel オブジェクトが管理するオブジェクトを表現するクラスの一部を説明します。以下で説明するオブジェクトは AwModel オブジェクトから得ることができます。

AwColorAssignment	色割り付け表
AwDrafParam	製図定数
AwGeomParam	幾何演算定数
AwItemAttrTable	アイテム属性バンドルテーブル
AwMiscParam	その他のパラメータ
AwModelTitleSet	モデルタイトルセット
AwSelectableMask	アイテム選択マスク

下記はこの章で解説する主なクラスを含むクラス図です。



8.1　AwGeomParam クラス

このクラスは幾何演算定数を表現します。AwGeomParam オブジェクトは AwModel オブジェクトから得ることができます。

- 演算定数

- デフォルト半径（円・円弧）
- 曲線オフセットコマンドのパラメータ
- テンポラリポイント作成メソッド
- 円・円弧、Cubic Bezier 曲線のスクリーン表示精度
- 円・円弧、Cubic Bezier 曲線のプリント・プロット精度

8.1.1 アクセッサ

最短長を取得／設定する。この値は曲線の最短長、円の最小半径、二点を同一とみなす許容差などを意味します。

```
double GetMinLength() const;
bool   SetMinLength(double len);
```

最短長 (1.0e-8 <= len)。

円の半径を取得／設定する。円作成コマンドなどで半径が指定されないときに使用するデフォルト値です。

```
double GetRadius() const;
bool   SetRadius(double radius);
```

円の半径 (GetMinLength() < radius)。

8.1.2 テンポラリポイント作成メソッド

テンポラリポイント作成メソッドの操作をするメソッドがあります。テンポラリポイント作成メソッドには 2 つのグループがあります。

グループ 1

AwGeomParam::TPNT_END	端点
AwGeomParam::TPNT_DIG	ディジタイズ点
AwGeomParam::TPNT_MID	中間点
AwGeomParam::TPNT_CEN	円中心点
AwGeomParam::TPNT_INT	交点
AwGeomParam::TPNT_AUTO	自動点
AwGeomParam::TPNT_NODE	ノード
AwGeomParam::TPNT_DIST	端点から指定した距離の点
AwGeomParam::TPNT_POS	APG, SYMBOL, SUBMODEL 配置点
AwGeomParam::TPNT_DYN	自動点 2

グループ 2

AwGeomParam::TPNT_PRJ	投影点
AwGeomParam::TPNT_VEC	ベクトル点

グループ 1 は必ずどれかひとつを選択しなければなりません。デフォルト値は TPNT_AUTO です。
グループ 2 は投影点かベクトル点または指定なしです。デフォルト値は指定なしです。

テンポラリポイント作成メソッドを設定します。

```
void SetPointMethod(int type);
```

type　ポイント作成メソッド。TPNT_PRJ か TPNT_VEC はグループ 2 のメソッドに設定します。それ以外はグループ 1 のメソッドに設定し、グループ 2 のメソッドをクリアします。

テンポラリポイント作成メソッドを変更します。

```
void SetPointMethod(int action, int type);
```

action　変更するものを指定する整数。

0 : 引数 type が TPNT_PRJ か TPNT_VEC はグループ 2 のメソッドに設定します。それ以外はグループ 1 のメソッドに設定し、グループ 2 のメソッドをクリアします。SetPointMethod(int type) メソッドと同じです。

1 : 引数 type が TPNT_PRJ か TPNT_VEC はグループ 2 のメソッドに設定します。それ以外はグループ 1 のメソッドに設定しますが、グループ 2 のメソッドは変更しません。
2 : グループ 2 のメソッドが引数 type と同じであればグループ 2 のメソッドをクリアします。引数 type は TPNT_PRJ か TPNT_VEC。
3 : グループ 2 のメソッドが引数 type と同じであればグループ 2 のメソッドをクリアします。そうでなければ。引数 type をグループ 2 のメソッドに設定します。引数 type は TPNT_PRJ か TPNT_VEC。

type　ポイント作成メソッド。

テンポラリポイント作成メソッドを得ます。
```
int GetPointMethod() const;
```
戻り値　グループ 1 のメッソッドを返します。
```
int GetPointMethod2() const;
```
戻り値　グループ 2 のメッソッドを返します。TPNT_PRJ、TPNT_VEC または指定なし (=0) です。

最後に記憶したポイント作成メソッドに戻します。
```
void RestorePointMethods();
```
現在のポイント作成メソッドを記憶します。
```
void SavePointMethods();
```

8.2 AwDrafParam クラス

このクラスは製図定数を表現します。AwDrafParam オブジェクトは AwModel オブジェクトから得ることができます。

8.2.1 初期値設定

このオブジェクトに初期値を設定します。
```
int SetDefaults(int munit, int istd);
```
munit　モデルの長さの単位 (1 : mm, 4 : Inch)。
istd　製図基準 (3 : ANSI, 7 : JIS)。
戻り値　成功したら 0 を返します。

寸法サブレコードが持つパラメータをこのオブジェクトに設定します。
```
void Set(const AwSrDimension& srDimension);
```
srDimension　AwSrDimension オブジェクト。

グラフィックステキストサブレコードが持つパラメータをこのオブジェクトに設定します。
```
void Set(const AwSrText& srTextParam);
```
srDimension　AwSrText オブジェクト。

8.2.2 テキストパラメータ

ASCII テキストフォント番号を取得／設定します。
```
int  GetAsciiFontId() const;
void SetAsciiFontId(int id);
```
1-99 = 英数字ストロークフォント
102-109 = アウトラインフォントの英数字部分を使う
111-130 = トゥルータイプフォントの英数字部分を使う (OS 依存、OS 間の互換性なし)

日本語テキストフォント番号を取得／設定します。
```
int  GetJapaneseFontId() const;
void SetJapaneseFontId(int id);
```

101 = 日本語ストロークフォント
102-109 = アウトラインフォント
111-130 = トゥルータイプフォント (OS 依存、OS 間の互換性なし)

文字高さを取得／設定します。
```
double  GetCharHeight() const;
void    SetCharHeight(double height);
```
ドローイングレイアウトスペースでの文字高さ (0.01 - 327.67)。

文字列角度を取得／設定します。
```
double  GetTextAngle() const;
void    SetTextAngle(double angle)
```
角度 (0 － 360.0 度)。

文字列表示モードを取得／設定します。
```
int  GetRotateInternalText() const;
void SetRotateInternalText(int flag);
```
0= 横書き、1= 縦書き

X 座標ミラーフラッグを取得／設定します。
```
int  GetTextMirrorX() const;
void SetTextMirrorX(int flag);
```
0= しない、1= テキストの Y 軸に対して反転する。

Y 座標ミラーフラッグを取得／設定します。
```
int  GetTextMirrorY() const;
void SetTextMirrorY(int flag);
```
0= しない、1= テキストの X 軸に対して反転する。

文字傾き角を取得／設定します。
```
   double GetCharSlantAngle() const;
   void   SetCharSlantAngle(double angle);
```
角度 (-85.00 ～ 85.00、0.01 刻み)。

文字縦横比を取得／設定します。
```
double  GetCharAspectRatio() const;
void    SetCharAspectRatio(double ratio);
```
文字高さを基準とした比率 (= 幅 / 高さ)。(0.1 - 10.0)。

文字列表示水平基準を取得／設定します。
```
int  GetTextJustification() const;
void SetTextJustification(int code);
```
BASE_LEFT= 左詰め、BASE_CENTER= 中央、BASE_RIGHT= 右詰め。

行幅整列係数を取得／設定します。
```
int  GetWidthFill() const;
void SetWidthFill(int code);
```
0 ～ 50% (1% 刻み), 55% ～ 100% (5% 刻み)。

水平方向原点基準を取得／設定します。
```
int  GetTextHorizontalOrigin() const;
void SetTextHorizontalOrigin(int code);
```
BASE_LEFT= 左、BASE_CENTER= 中央、BASE_RIGHT= 右。

垂直方向原点基準を取得／設定する。
```
int  GetTextVerticalOrigin() const;
void SetTextVerticalOrigin(int code);
```

BASE_BOTTOM= 下、BASE_CENTER= 中央、BASE_TOP= 上。

水平方向ゆとり幅を取得／設定します。
```
double GetTextboxHorizontalMargin() const;
void   SetTextboxHorizontalMargin(double margin);
```
ドローイングレイアウトスペースでのゆとり幅 (0.0 ～ 327.67、0.01 刻み)。

垂直方向ゆとり幅を取得／設定します。
```
double GetTextboxVerticalMargin() const;
void   SetTextboxVerticalMargin(double margin);
```
ドローイングレイアウトスペースでのゆとり幅 (0.0 ～ 327.67、0.01 刻み)。

文字間隔を取得／設定します。
```
double GetInterCharSpaceRatio() const;
void   SetInterCharSpaceRatio(double ratio);
```
文字高さを基準とした比率 (= 文字間隔 / 高さ)。(0.0 - 10.0)。

行間隔を取得／設定します。
```
double GetInterLineSpaceRatio() const;
void   SetInterLineSpaceRatio(double ratio);
```
文字高さを基準とした比率 (= 行間隔 / 高さ)。(0.0 - 10.0)。

文字列枠／下線表示スタイルを取得／設定します。
```
int    GetTextboxStyle() const;
void SetTextboxStyle(int code);
```
STYLE_NONE= なし、STYLE_BOX= 枠、STYLE_UNDERLINE= 下線、
STYLE_BOX_UNDERLINE= 枠と下線、STYLE_VARIABLE_LINE= 可変長下線、
STYLE_DOUBLE_LINE = 二重下線。

8.2.3　寸法値パラメータ

製図基準を取得／設定します。
```
int  GetDimensionTextAlignment() const;
void SetDimensionTextAlignment(int value);
```
0 = 水平／寸法線分断 (ANSI)
1 = 平行／寸法線閉 (JIS)

十進整数部の 3 桁区切り記号を取得／設定します。
```
int  GetThousandsSeparator() const;
void SetThousandsSeparator(int value);
```
0 = なし、1 = カンマ、2 = 空白、3 = ピリオド。

十進小数点記号を取得／設定します。
```
int  GetDecimalSymbol() const;
void SetDecimalSymbol(int value);
```
0 = ピリオド、1 = カンマ

十進表示の小数部の「後ろのゼロを除去」を取得／設定します。
```
int  GetTrailZerosSuppression() const;
void SetTrailZerosSuppression(int value);
```
0 = 後ろのゼロを除去する、1= 後ろのゼロも表示する。

8.2.4　長さ寸法の値のパラメータ

単寸法／両寸法モードを取得／設定します。

```
int  GetDualDimensionTexts() const;
void SetDualDimensionTexts(int value);
```
0 = 単一、1 = mm/inch 併記、2 = inch/mm 併記。

寸法値表示形式を取得／設定します。
```
int  GetFractionType() const;
void SetFractionType(int value);
```
0 = 十進、1 = feet & inch、2 = feet、3 = inch。

寸法値単位を取得／設定します。
```
int  GetUnit() const;
void SetUnit(int value);
```
1 = milimeter、2 = centimeter、3 = meter
4 = inch、5 = feet、6 = mil (1/1000 inch)

単位記号表示を取得／設定します。
```
int  GetUnitDisplay() const;
void SetUnitDisplay(int value);
```
0 = 表示しない、1 = シンボル、2 = 省略文字。

インチ分数の単位を取得／設定します。
```
int  GetFractionPrecision() const;
void SetFractionPrecision(int value);
```
寸法値表示形式が十進でないときだけ参照する。
0 = 1/1、1 = 1/2、2 = 1/4、3 = 1/8、4 = 1/16、5 = 1/32、6 = 1/64

十進表示の小数桁数を取得／設定します。
```
int  GetDecimalPrecision() const;
void SetDecimalPrecision(int value);
```
0 － 12。寸法値表示形式が十進のときだけ参照する。

半径の寸法補助記号と位置を取得／設定します。
```
int  GetRadiusSymbolPosition() const;
void SetRadiusSymbolPosition(int value);
```
0 = 前 R、1 = 後 R、2 = なし。

直径の寸法補助記号と位置を取得／設定します。
```
int  GetDiameterSymbolPosition() const;
void SetDiameterSymbolPosition(int value);
```
0 = φ 前、1 = 後 φ、2 = DIA 前、3 = 後 DIA
4 = φ 前、5 = 後 φ、6 = DIA 前、7 = 後 DIA
0 ～ 3 は直径寸法と長さ寸法の φ 追加の両方に使用する。
4 ～ 7 は直径寸法に寸法補助記号を付けない。長さ寸法の φ 追加にだけ使用する。

長さ寸法値の倍率を取得／設定します。
```
double GetDimensionScale() const;
void SetDimensionScale(double scale);
```
倍率 (1.0e-8 <= scale)。

8.2.5　角度寸法の値のパラメータ

寸法値の表示形式を取得／設定します。
```
int  GetAngleUnitDisplay() const;
void SetAngleUnitDisplay(int value);
```
0 = 度分秒 (60 進)
1 = 十進、単位表示 (Deg)

2 = 十進、単位表示なし
3 = 十進、単位表示 (° シンボル)

度分秒 (60 進) で表示する最小単位を取得／設定します。
```
int  GetAngleUnit() const;
void SetAngleUnit(int value);
```
1 = 度まで、2 = 分まで、3 = 秒まで。

十進表示の小数桁数を取得／設定します。
```
int  GetAngleDecimalPrecision() const;
void SetAngleDecimalPrecision(int value);
```
小数桁数は 0 - 12。

8.2.6　寸法許容差のパラメータ

十進表示の小数桁数を取得／設定します。
```
int  GetToleranceDecimalPrecision() const;
void SetToleranceDecimalPrecision(int value);
```
小数桁数は 1 - 12。

十進表示の小数部の「後ろのゼロを除去」を取得／設定します。
```
int  GetToleranceTrailZerosSuppression() const;
void SetToleranceTrailZerosSuppression(int value);
```
0 = 後ろのゼロを除去する、1= 後ろのゼロも表示する。

上／下の寸法許容差の文字高さを取得／設定します。
```
double GetToleranceCharHeight() const;
void   SetToleranceCharHeight(double height);
```
ドローイングレイアウトスペースでの文字高さ (0.01 - 327.67)。

± 寸法許容差の文字高さを取得／設定します。
```
double GetToleranceCharHeight2() const;
void   SetToleranceCharHeight2(double height);
```
ドローイングレイアウトスペースでの文字高さ (0.01 - 327.67)。

上の寸法許容差のデフォルト値を取得／設定します。
```
const std::string& GetUpperTolerance() const;
void  SetUpperTolerance(const std::string& value);
```
32 バイト以下。

下の寸法許容差のデフォルト値を取得／設定します。
```
const std::string& GetLowerTolerance() const;
void  SetLowerTolerance(const std::string& value);
```
32 バイト以下。

± 寸法許容差のデフォルト値を取得／設定します。
```
const std::string& GetBilateralTolerance() const;
void  SetBilateralTolerance(const std::string& value);
```
32 バイト以下。

8.2.7　寸法線、寸法補助線パラメータ

寸法線の矢の種類を取得／設定します。
```
int  GetDimensionArrowhead() const;
void SetDimensionArrowhead(int type);
```

矢の種類 (0= なし、1 ～ n= マーク番号)。

寸法線の矢の大きさを取得／設定します。
```
double GetDimensionArrowheadSiz() const;
void SetDimensionArrowheadSize(double size);
```
ドローイングレイアウトスペースでの大きさ (0.01 - 327.67)。

寸法線スタブの長さを取得／設定します。
```
double GetLeaderStubLength() const;
void SetLeaderStubLength(double length);
```
ドローイングレイアウトスペースでの長さ (0.01 - 327.67)。

寸法補助線ギャップを取得／設定します。
```
double GetWitnessLineGap() const;
void SetWitnessLineGap(double length);
```
ドローイングレイアウトスペースでの長さ (0.01 - 327.67)。

寸法補助線オーバーシュートを取得／設定します。
```
double GetWitnessLineOvershoot() const;
void SetWitnessLineOvershoot(double length);
```
ドローイングレイアウトスペースでの長さ (0.01 - 327.67)。

8.2.8 その他の寸法パラメータ

並列寸法の間隔を取得／設定します。
```
double GetInterdimensionSpace() const;
void SetInterdimensionSpace(double space);
```
ドローイングレイアウトスペースでの間隔 (0.01 - 327.67)。

半径寸法の内側引出し線の長さを取得／設定します。
```
double GetRadiusLineLength() const;
void SetRadiusLineLength(double length);
```
ドローイングレイアウトスペースでの長さ (0.01 - 327.67)。

累進寸法の起点記号を取得／設定します。
```
int GetBaselineSymbol() const;
void SetBaselineSymbol(int type);
```
起点記号の種類 (0= なし、1 ～ n= マーク番号)。

8.2.9 リファレンスノート、マークおよび引出線パラメータ

風船のマーク番号を取得／設定します。
```
int GetRflFrame() const;
void SetRflFrame(int type);
```
風船の種類 (0= なし、1 ～ n= マーク番号)。

風船のマークの大きさを取得／設定します。
```
double GetRflFrameSize() const;
void SetRflFrameSize(double size);
```
ドローイングレイアウトスペースでの大きさ (0.01 - 327.67)。

風船の引出し線の矢の種類を取得／設定します。
```
int GetRflArrrowhead() const;
void SetRflArrrowhead(int type);
```
矢の種類 (0= なし、1 ～ n= マーク番号)。

風船の引出し線の矢の大きさを取得／設定します。
```
double GetRflArrrowheadSize() const;
void SetRflArrrowheadSize(double size);
```
ドローイングレイアウトスペースでの大きさ (0.01 - 327.67)。

風船の文字の大きさを取得／設定します。
```
double GetRflCharHeight() const;
void SetRflCharHeight(double height);
```
ドローイングレイアウトスペースでの大きさ (0.01 - 327.67)。

面の指示記号の大きさを取得／設定します。
```
double GetSurfaceTextureSymbolSize() const;
void SetSurfaceTextureSymbolSize(double size);
```
ドローイングレイアウトスペースでの大きさ (0.01 - 327.67)。

面の指示記号の文字高さを取得／設定します。
```
double GetSurfaceTextureCharHeight() const;
void SetSurfaceTextureCharHeight(double height);
```
ドローイングレイアウトスペースでの大きさ (0.01 - 327.67)。

溶接記号の大きさを取得／設定します。
```
double GetWeldSymbolSize() const;
void SetWeldSymbolSize(double size);
```
ドローイングレイアウトスペースでの大きさ (0.01 - 327.67)。

溶接記号の文字高さを取得／設定します。
```
double GetWeldCharHeight() const;
void SetWeldCharHeight(double height);
```
ドローイングレイアウトスペースでの大きさ (0.01 - 327.67)。

引出し線の最初の線分の丸め角度を取得／設定します。
```
int GetLeaderRoundoffAngle() const;
void SetLeaderRoundoffAngle(int angle);
```
丸め角度 (0 - 180 度)。

引出し線の最後の線分の丸め角度を取得／設定します。
```
double GetLeaderRoundoffAngle2() const;
void SetLeaderRoundoffAngle2(double angle);
```
丸め角度 (0 - 180 度)。

引出し線の矢の種類を取得／設定します。
```
int GetLeaderArrowhead() const;
void SetLeaderArrowhead(int type);
```
矢の種類 (0= なし、1 ～ n= マーク番号)。

引出し線の矢の大きさを取得／設定します。
```
double GetLeaderArrowheadSize() const;
void SetLeaderArrowheadSize(double size);
```
ドローイングレイアウトスペースでの大きさ (0.01 - 327.67)。

8.2.10 幾何公差パラメータ

幾何公差記入枠の行の高さを取得／設定します。
```
double GetGtRowHeight() const;
void SetGtRowHeight(double height);
```
ドローイングレイアウトスペースでの大きさ (0.01 - 327.67)。

幾何公差記号の大きさを取得／設定します。
```
double GetGtSymbolSize() const;
void SetGtSymbolSize(double size);
```
ドローイングレイアウトスペースでの大きさ (0.01 - 327.67)。

幾何公差の引出し線の矢の種類を取得／設定します。
```
int GetGtArrrowhead() const;
void SetGtArrrowhead(int type);
```
矢の種類 (0= なし、1 ～ n= マーク番号)。

幾何公差の引出し線の矢の大きさを取得／設定します。
```
double GetGtArrrowheadSize() const;
void SetGtArrrowheadSize(double size);
```
ドローイングレイアウトスペースでの大きさ (0.01 - 327.67)。

データム文字記入枠の種類を取得／設定します。
```
int GetGtDatumFrame() const;
void SetGtDatumFrame(int type);
```
文字記入枠の種類 (0= なし、1 ～ n= マーク番号)。

データム文字の高さを取得／設定します。
```
double GetGtDatumCharHeight() const;
void SetGtDatumCharHeight(double height);
```
ドローイングレイアウトスペースでの大きさ (0.01 - 327.67)。

データムの引出し線の矢の種類を取得／設定します。
```
int GetGtDatumArrowhead() const;
void SetGtDatumArrowhead(int type);
```
矢の種類 (0= なし、1 ～ n= マーク番号)。

8.2.11 切断線のパラメータ

切断線矢印の種類を取得／設定します。
```
int GetCuttingLineArrowhead() const;
void SetCuttingLineArrowhead(int type);
```
切断線矢印の種類 (0= なし、1 ～ n= マーク番号)。

切断線矢印の大きさを取得／設定します。
```
double GetCuttingLineArrowheadSize() const;
void SetCuttingLineArrowheadSize(double size);
```
ドローイングレイアウトスペースでの大きさ (0.01 - 327.67)。

切断線の文字高さを取得／設定します。
```
double GetCuttingLineCharHeight() const;
void SetCuttingLineCharHeight(double height);
```
ドローイングレイアウトスペースでの大きさ (0.01 - 327.67)。

切断線の線の表示を取得／設定します。
```
int GetCuttingLineDisplay() const;
void SetCuttingLineDisplay(int mode);
```
0= なし、1= 表示、2= 折れ曲がり部のみ表示。

8.2.12 作表のパラメータ

表の行の高さを取得／設定します。

```
double GetTableRowHeight() const;
void SetTableRowHeight(double height);
```
ドローイングレイアウトスペースでの大きさ (0.01 - 327.67)。

表の文字高さを取得／設定します。
```
double GetTableCharHeight() const;
void SetTableCharHeight(double height);
```
ドローイングレイアウトスペースでの大きさ (0.01 - 327.67)。

表の列の幅を取得／設定します。
```
int GetTableColumnLength() const;
void SetTableColumnLength(int chars);
```
1 - 255 バイト。

十進表示の小数桁数を取得／設定します。
```
int GetTableDecimalPrecision() const;
void SetTableDecimalPrecision(int digits);
```
小数桁数　0 - 12。

面積作表の単位変換定数を取得／設定します。
```
double GetAreaScale() const;
void SetAreaScale(double scale);
```
単位変換定数 (0 < scale)。

8.2.13 円中心線のパラメータ

中心線のオーバーシュートの指定方法を取得／設定します。
```
int GetCenterLineOvershootType() const;
void SetCenterLineOvershootType(int type);
```
0= モデルスペースでの長さ、1= ドローイングレイアウトスペースでの長さ、2= 半径の比率。

中心線のオーバーシュートの長さを取得／設定します。
```
double GetCenterLineOvershootSize() const;
void SetCenterLineOvershootSize(double length);
```
0= モデルスペースでの長さ
1= ドローイングレイアウトスペースでの長さ
2= 半径の比率 (-0.5 ～ 0.5)。

8.3 AwMaskSet 抽象クラス

このクラスはアイテムの表示や選択を制御するマスクセットを表現する抽象クラスです。マスクセットはアイテム属性ごとの５種類のマスクを持ちます。

AwItemAttributes::BY_ITEMTYPE	アイテム・タイプ
AwItemAttributes::BY_CLASS	アイテム・クラス
AwItemAttributes::BY_REVISION	アイテム・レビジョン
AwItemAttributes::BY_LINEFONT	アイテム線種
AwItemAttributes::BY_LINEWEIGHT	アイテム線幅

マスクを初期化します。全種類のマスクの値を true にします。
```
void SetDefault();
```

マスクを設定します。
```
void Set(const AwMaskSet& maskset);
```
maskset　ソースマスクセットオブジェクト。
```
void Set(int type, const AwMaskSet& maskset);
```

maskset　ソースマスクセットオブジェクト。
type　設定するマスクの種類を示す整数。

8.4 AwSelectableMask クラス

このクラスはアイテムの選択を制御するマスクセットを表現します。このクラスは AwMaskSet クラスの派生クラスです。
このクラスはアイテム選択マスクとして使用しており、それらは AwModel オブジェクトから得ます。
また AwPermDbIterator クラスは探索対象アイテムを選択するのにこのクラスを使用しています。

8.4.1 Getter

指定したアイテムが選択可能か判定します。
```
bool IsSelectable(const AwItemAttributes& attr) const;
```
attr　検査する AwItemAttributes オブジェクト。
戻り値　AwItemAttributes オブジェクトのアイテムタイプ、クラス、レビジョン、線種、線幅の全てが選択可能なら true を返します。

指定したマスクの要素が選択可能か判定します。
```
bool IsSelectable(int type, int no) const;
```
type　マスクの種類を示す整数。
no　要素の番号 (1-)。
戻り値　指定したマスクの要素が選択可能なら true を返します。

指定したマスクの全ての要素が選択可能か判定します（デフォルトの状態)。
```
bool IsSelectableAll(int type) const;
```
type　マスクの種類を示す整数。
戻り値　指定したマスクの全ての要素が選択可能なら true を返します。

8.4.2 Setter

指定したマスクのひとつの要素の値を設定します。
```
void SetSelectable(int type, int no, bool value);
```
type　マスクの種類を示す整数。
no　設定する要素の番号 (1-)。
value　選択するなら true、選択しないなら false。

指定したマスクの連続する複数の要素の値を設定します。
```
void SetSelectables(int type, int from, int to, bool value);
```
type　マスクの種類を示す整数。
from　設定する最初の要素番号 (1-)。
to　設定する最後の要素番号 (1 <= from <= to)。
value　選択するなら true、選択しないなら false。

指定したマスクの全ての要素の値を設定します。
```
void SetSelectableAll(int type, bool value);
```
type　マスクの種類を示す整数。
value　選択するなら true、選択しないなら false。

8.4.3 アイテム選択マスク

アイテム選択マスクにはテンポラリなマスクとパーマネントなマスクの二つがあります。どちらも AwSelectableMask オブジェクトです。以下にパーマネントなマスクの例を示しますが、テンポラリなマスクも同様です。

例）クラス 5 も選択可にする。

```
AwSelectableMask* pMask = pModel->GetSelectableMask();
if (pMask)
  pMask->SetSelectable(AwItemAttributes::BY_CLASS, 5, true);
```

例) 選択クラスを 1 から 100 と 201 から 253 と 255 にする。また製図アイテムと点アイテムを選択しないようにする。

```
AwSelectableMask* pMask = pModel->GetSelectableMask();
if (pMask == NULL)
  return;
// CLS/SEL REL ALL ADD 1 -100 201 -253 255 <CE>
pMask->SetSelectableAll(AwItemAttributes::BY_CLASS, false);          // 全クラス
pMask->SetSelectables(AwItemAttributes::BY_CLASS, 1, 100, true);     //クラス 1 - 100
pMask->SetSelectables(AwItemAttributes::BY_CLASS, 201, 253, true);  //クラス 201 - 253
pMask->SetSelectable(AwItemAttributes::BY_CLASS, 255, true);         // クラス 255
// ITM/SEL ADD MANY REL MDRF MPNT <CE>
pMask->SetSelectableAll(AwItemAttributes::BY_ITEMTYPE, true);         // 全タイプ
pMask->SetSelectable(AwItemAttributes::BY_ITEMTYPE, AwItem::POINT, false);
pMask->SetSelectable(AwItemAttributes::BY_ITEMTYPE, AwMaskSet::MASK_DRAFTING, false); // 製図アイテム
```

テンポラリ選択マスクを解除するには次のようにします。テンポラリなマスクの値をパーマネントなマスクと同じにします。

```
const AwSelectableMask* pSelMask = pModel->GetSelectableMask();
AwSelectableMask* pTmpMask       = pModel->GetTemporarySelectableMask();
if (pSelMask && pTmpMask)
  pTmpMask->Set(*pSelMask);
```

テンポラリ選択マスクをパーマネント選択マスクにするには次のようにします。

```
AwSelectableMask* pSelMask       = pModel->GetSelectableMask();
const AwSelectableMask* pTmpMask = pModel->GetTemporarySelectableMask();
if (pSelMask && pTmpMask)
  pSelMask->Set(*pTmpMask);
```

クラス選択マスクだけを設定したいなら次のようにします。

```
pSelMask->Set(AwItemAttributes::BY_CLASS, *pTmpMask);
```

8.5 AwModelTitleSet クラス

このクラス はモデルタイトル全般の情報を持ち、モデルタイトル AwModelTitle オブジェクトを管理 (所有）します。
モデルタイトルは定義ファイルに記述します。ユーザ定義のモデルタイトル は 1 － 200 番です。201 番以降はアプリケーションが使用します。詳細は『システム管理者の手引き』の『モデルタイトル』の章を参照してください。
このクラスは定義ファイルを読込み、モデルタイトルオブジェクトを生成します。タイトルオブジェクトはタイトル表示オーダー、タイトル識別番号で昇順に並べます。

AwModelTitleSet オブジェクトは AwModel オブジェクトから得ます。

8.5.1 タイトルを得る

タイトルオブジェクトを得ます。
```
const AwModelTitle* GetTitle(int tid) const;
AwModelTitle* GetTitle(int tid);
```
tid　　タイトル番号。1 <= tid。
戻り値　AwModelTitle オブジェクトへのポインタを返します。指定のタイトル番号のオブジェクトが存在しないときは null を返します。

8.6 AwModelTitle クラス

このクラス はモデルタイトルを表現します。
AwModelTitle オブジェクトは AwModelTitleSet オブジェクトから得ます。

8.6.1 定数

アプリケーションが定義するタイトルの番号

定数名	説明
ID_MAIN_TITLE	主タイトル
ID_MODELFILE	アクティブモデルファイル名
ID_CREATION_DATE	モデルファイル作成日付
ID_CREATION_TIME	モデルファイル作成時刻
ID_MODIFIED_DATE	モデルファイル更新日付
ID_MODIFIED_TIME	モデルファイル更新時刻
ID_USER_NAME	アカウント名
ID_DRAWING_STANDARD	SXF 用
ID_DRAWING_TYPE	SXF 用

8.6.2 タイトル値の取得／設定

タイトル値を変更できるのはユーザ定義のタイトル（番号 は 1 - 200) と ID_MAIN_TITLE です。それ以外のアプリケーション定義タイトルは変更しないでください。ID_MODELFILE から ID_MODIFIED_-TIME はアクティブモデルに関するもので変更してはいけません。

タイトル値を得ます。
```
const std::string& GetValue() const;
```
戻り値　このオブジェクトのタイトル値を返します。

タイトル値を設定します。引数 text の値が適切であればタイトル値を更新します。
```
int  CheckAndSetValue(const std::string& text);
```
text　　タイトル値。
戻り値　タイトル値を更新したとき 0 を返します。
2 :モデルタイトルの文字数が規定より短い。
3 ： モデルタイトルの文字数が規定より長い。
4 ： モデル名が長すぎる（32 バイト以内)。ID_MODELFILE のみ。

5 ： 不正な文字を含んでいる。

タイトル値を空にします。
```
void RemoveValue();
```

8.7 AwItemAttrTable クラス

このクラスはコマンド・カテゴリごとのデフォルトアイテム属性を持ちます。事前にコマンド・カテゴリごとのアイテム属性を設定しておきます。コマンドが切り替わったら、そのコマンドが属するカテゴリに対応するアイテム属性を現在のアイテム属性に設定するようにします。こうすることでを現在のアイテム属性の変更を省略できます。対象となるアイテム属性はクラス、レビジョン、線種、線幅の 4 種類です。詳細は『コマンド・リファレンス』の『線種線幅定数』を参照してください。

AwItemAttrTable オブジェクトは AwModel オブジェクトから得ます。

8.7.1 定数

コマンド・カテゴリ

コマンド・カテゴリ	作成するアイテム
GEOMETRY	点、線分、円、ストリング、スプライン
DIMENSION	寸法
TEXT	注記
MARK	マーク
REFLABEL	リファレンスラベル／ノート
GT	幾何公差
CUTPLANE	切断線
CENTERLINE	円中心線
HATCHING	ハッチング
COMPOSITE	コンポジットアイテム
APG	パラメトリックアイテム
PARTS	パーツ
SUBMODEL	サブモデル

8.7.2 バンドル値の取得／設定

指定したカテゴリのアイテム属性の値を得ます。
```
int GetValueAt(int categ, int type) const;
```
categ　　カテゴリを表す整数。
type　　アイテム属性の種類を表す整数 (AwItemAttributes::BY_CLASS - BY_LINEWEIGHT)。
戻り値　　アイテム属性のデフォルト値を返します。エラーの時は 0 を返します。

アイテム属性取得モードを得ます。
```
int GetModeAt(int categ, int type) const;
```
categ　　カテゴリを表す整数。

type　　アイテム属性の種類を表す整数 (AwItemAttributes::BY_CLASS - BY_LINEWEIGHT)。
戻り値　　取得モードとデフォルト値を結合したもの返します。

アイテム属性取得モードを設定します。
```
bool SetModeAt(int categ, int type, int mode);
```
categ　　カテゴリを表す整数。
type　　アイテム属性の種類を表す整数 (AwItemAttributes::BY_CLASS - BY_LINEWEIGHT)。
mode　　属性取得モードとデフォルト値を結合したもの。
戻り値　　成功したら true を返します。

8.8 AwMiscParam クラス

このクラスはモデルの未分類の（あるいはその他の）パラメータを持ちます。

AwMiscParam オブジェクトは AwModel オブジェクトから得ます。

モデルの長さ単位を得ます。
```
int GetModelUnit() const;
```
戻り値　　モデルの長さ単位を表す整数を返します。

コード	単位の名称	略号	略号
1	Millimeter	mm	MM
2	Centimeter	cm	CM
3	Meter	m	M
4	Inch	in	IN
5	Feet	ft	FT
6	Mil	mil	MIL

8.9 AwColorAssignment クラス

このクラスはアイテムをスクリーンに表示するのに使用する色割り付け表を表現します。色割り付けタイプは下記に示すもので、このなかからひとつを選択します。色割り付けタイプをアイテムタイプとしていれば、アイテムタイプの色割り付け表を参照し色番号を決めます。

AwItemAttributes::BY_ITEMTYPE　　アイテムタイプ
AwItemAttributes::BY_CLASS　　アイテムクラス
AwItemAttributes::BY_REVISION　　アイテムレビジョン
AwItemAttributes::BY_LINEFONT　　アイテム線種
AwItemAttributes::BY_LINEWEIGHT　　アイテム線幅

AwColorAssignment オブジェクトは AwModel オブジェクトから得ることができます。

8.9.1 アクセッサ

現在の色割り付けタイプを取得／設定します。デフォルト値は BY_CLASS です。
```
int  GetAssignmentType() const;
void SetAssignmentType(int type);
```
タイプ (AwItemAttributes::BY_ITEMTYPE - AwItemAttributes::BY_LINEWEIGHT)。

8.9.2 色割り付け表

指定した AwItemAttributes に割り付けた色番号を得ます。
```
int  GetColor(const AwItemAttributes& attr) const;
int  GetColor(int type, const AwItemAttributes& attr) const;
```
type　色割り付けタイプ (AwItemAttributes::BY_ITEMTYPE - BY_LINEWEIGHT)。
attr　AwItemAttributes オブジェクト。
戻り値　色番号を返します。

指定した AwSrAttributes に割り付けた色番号を得ます。
```
int  GetColor(const AwSrAttributes& srAttr) const;
int  GetColor(int type, const AwSrAttributes& srAttr) const;
```
type　色割り付けタイプ (AwItemAttributes::BY_ITEMTYPE - BY_LINEWEIGHT)。
srAttr　AwSrAttributes オブジェクト。
戻り値　色番号を返します。

指定した要素に割り付けた色番号を得ます。
```
int  GetColor(int type, int no) const;
```
type　色割り付けタイプ (AwItemAttributes::BY_ITEMTYPE - BY_LINEWEIGHT)。
no　要素の番号 (1-)。たとえば type が BY_CLASS ならクラス番号です。
戻り値　色番号を返します。

以下のメソッドは色割り付けの設定を行ないますが、現在の色割り付けタイプは変更しません。
指定した色割り付けタイプのひとつの要素の色番号を設定します。
```
void SetColor(int type, int no, int color);
```
type　色割り付けタイプ (AwItemAttributes::BY_ITEMTYPE - BY_LINEWEIGHT)。
no　要素の番号 (1-)。たとえば type が BY_CLASS ならクラス番号です。
color　色番号 (1 - AwItemAttributes::MAX_ITEMCOLOR)。

指定した色割り付けタイプの全ての要素に同じ色番号を設定します。
```
void SetColors(int type, int color);
```
type　色割り付けタイプ (AwItemAttributes::BY_ITEMTYPE - BY_LINEWEIGHT)。
color　色番号 (1 - AwItemAttributes::MAX_ITEMCOLOR)。

例）クラス５の色を１番の色にします。それ以外は変更しません。
```
AwColourAssignment* pAssign = pModel->GetColorAssignment();
if (pAssign)
  pAssign->SetColor(AwItemAttributes::BY_CLASS, 5, 1);
```

8.10 関数

8.10.1 Radius001() 関数

半径値を設定します。この関数は半径を変更した後、ステータス領域にある半径値の表示を更新します。表示の更新が不要であれば AwGeomParam::SetRadius() メソッドを使用します。

```
int Radius001(double rad, int dspflg)
```
rad　設定する半径値。
dspfg　ステータス領域の半径値表示を更新するかどうか選択します。
　　0　更新しない。
　　1　更新する。
戻り値　半径を更新したら 0 を返します。

8.10.2 Clr002() 関数

現在の色割り付けの種類を設定します。この関数は色割り付けの種類を変更した後メニューやステータス領域の色の表示を更新します。このような表示の更新が不要であれば AwColorAssignment::SetAssignmentType() メソッドを使用します。

```
int Clr002(int type)
```

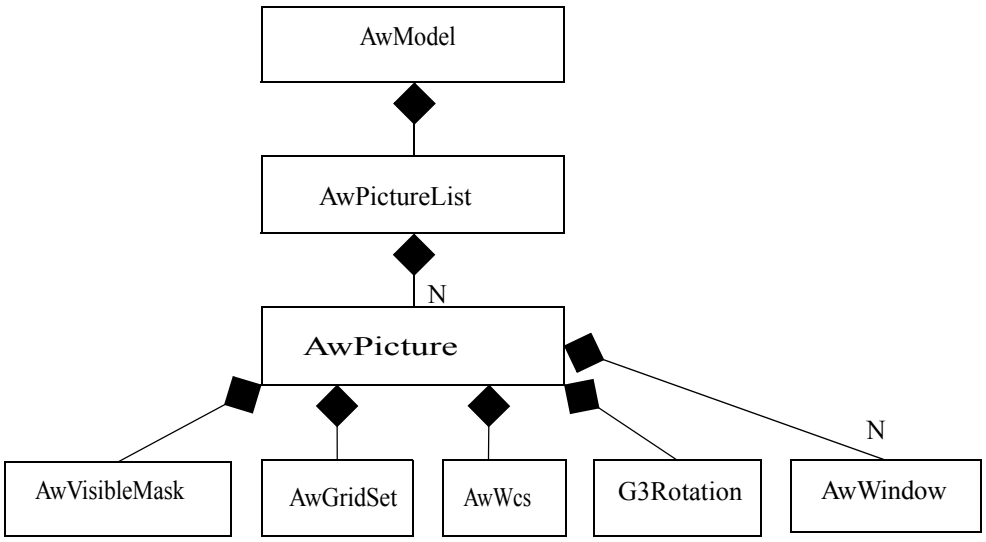
type　色割り付けの種類。

AwItemAttributes::BY_ITEMTYPE	アイテムタイプ
AwItemAttributes::BY_CLASS	クラス
AwItemAttributes::BY_REVISION	レビジョン
AwItemAttributes::BY_LINEFONT	線種
AwItemAttributes::BY_LINEWEIGHT	線幅

戻り値　色割り付けの種類を更新したら 0 を返します。

第 9 章　ピクチャ

この章ではアプリケーションモデル AwModel オブジェクトが管理するピクチャを説明します。下記はこの章で解説する主なクラスを含むクラス図です。



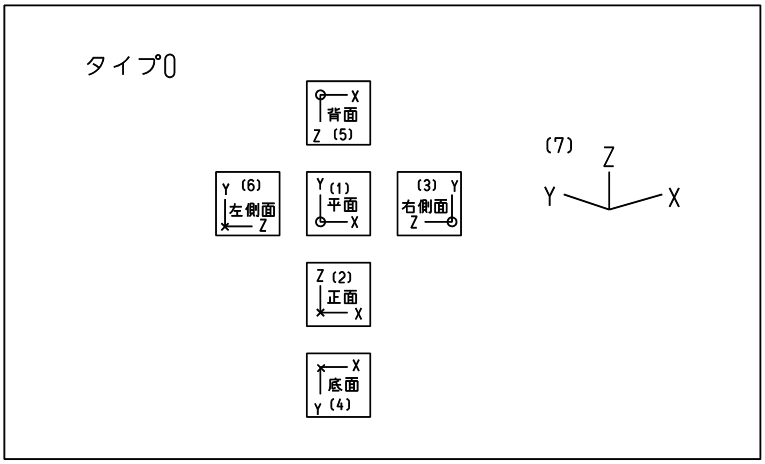
9.1　AwPictureList クラス

このクラスは AwPicture オブジェクトのコンテナです。ピクチャの共通情報を持ち、AwPicture オブジェクトを管理 (所有）します。
AwPictureList オブジェクトは AwModel オブジェクトから得ます。

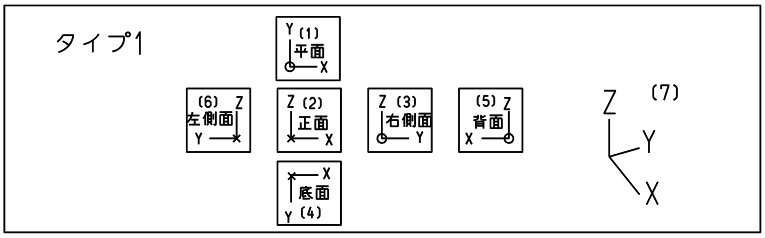
9.1.1　投影法

投影法を以下の二つから選択できます。指定しなければ投影法 #0 です。ピクチャ＃１から＃７には投影法に従った回転行列を設定します。

投影法 #0（デフォルト）



投影法 #1



投影法を取得／設定します。
```
int GetProjectionType() const;
void SetProjectionType(int type);
```
0 = Projection type #0
1 = Projection type #1

選択されている投影法に従ってピクチャの回転行列を設定します。ピクチャ＃１－＃７の回転行列を設定します。
```
void SetPictureRotations();
```

9.1.2 ピクチャを得る

指定した番号のピクチャを得ます。
```
const AwPicture* GetPicture(int pid) const;
AwPicture* GetPicture(int pid);
```
pid　　ピクチャの番号。
戻り値　ピクチャ番号が pid の AwPicture オブジェクトへのポインタを返します。オブジェクトが存在しないときは null ポインタを返します。

指定した名前のピクチャを得ます。
```
const AwPicture* GetPicture(const std::string& name) const;
```
name　　ピクチャの名前。
戻り値　ピクチャ名が name の AwPicture オブジェクトへのポインタを返します。オブジェクトが存在しないときは null ポインタを返します。

アクティブピクチャを得ます。
```
const AwPicture* GetActivePicture() const;
```

戻り値　アクティブピクチャの AwPicture オブジェクトへのポインタを返します。オブジェクトが存在しないときは null ポインタを返します。

指定した番号のピクチャを作成します。それが既にあるときは作成しません。
```
AwPicture* CreatePicture(int pid);
```
pid　　ピクチャ番号。
戻り値　ピクチャ番号が pid の AwPicture オブジェクトへのポインタを返します。ピクチャ番号の誤りなどでオブジェクトが作成できないときは null ポインタを返します。

アクティブピクチャの AwPicture オブジェクトを作成します。それが既にあるときは作成しません。
```
AwPicture* CreateActivePicture();
```
戻り値　アクティブピクチャの AwPicture オブジェクトへのポインタを返します。オブジェクトが作成できないときは null ポインタを返します。

9.2　AwPicture クラス

このクラスはピクチャを表現します。ピクチャは次の三つに区分します。
- **通常ピクチャ**
- **ドローイングピクチャ (SXF が必要とする)**
- **SXF 用特殊ピクチャ**

ドローイングピクチャはドローイングレイアウトにひとつずつ割り当てたピクチャです。ドローイングピクチャは用途が限定されているため、通常ピクチャに比べて制限があります。たとえば、ドローイングレイアウトに配置できるのは通常ピクチャに定義したウィンドウです。従ってドローイングピクチャにはウィンドウは定義できません。ドローイングピクチャはスケールは 1 で、座標はドローイングレイアウトスペースです。

以下の項目は通常ピクチャにのみ設定できます。
- **名前**（デフォルト値は空文字列）
- **説明文**（デフォルト値は空文字列）
- **スケール**（デフォルト値は 1）
- **回転行列**（デフォルト値は投影法に従った行列、または単位行列）
- ウインドウ

ピクチャは下記の情報も持っています。
- **表示マスク**（デフォルトは全て表示）
- **グリッド**（デフォルトはグリッドを適用しない状態）
- **補助座標**（デフォルトは補助座標を適用しない状態）

AwPicture オブジェクトは AwPictureList オブジェクトから得ます。

9.2.1　名前

このピクチャの名前を取得／設定します。ピクチャ名は重複してはなりません。

このピクチャの名前を得ます。
```
const std::string& GetName() const;
```

このピクチャに名前をつけます。このメソッドは名前の重複検査は行いません。
```
void  SetName(const std::string& name);
```

このピクチャの名前を除去します。
```
void RemoveName();
```

9.2.2 ピクチャスケール

ドローイングスケール が図面縮尺、ピクチャスケールが部分拡大図倍率に相当します。

文字やマークのピクチャ上の大きさはピクチャスケールとドローイングスケールを反映したものです。通常、ピクチャスケールとドローイングスケールはアイテム作成前に設定します。アイテム作成後にスケールを変更しても、それは文字やマークの大きさには反映しません。そのため、スケールの変更を記憶しておき、適切な時点で文字やマークの大きさを更新することになります。これを避けるにはアイテム作成後のスケールの変更は行わないことです。

このオブジェクトのピクチャスケールを得ます。
```
double GetScale() const;
```
戻り値　このオブジェクトのピクチャスケールを返します。

このオブジェクトのピクチャスケールを設定します。
```
bool SetScale(double scale, bool modified = true);
```
scale　　ピクチャスケール。0 < scale。
modified　ピクチャスケールの変更をマークするとき true、解除するとき false とします。

このオブジェクトのピクチャスケールが変更されているか調べます。
```
bool IsScaleModified() const;
```
戻り値　このオブジェクトのスケール変更がマークされていると true を返します。

9.2.3 回転行列

回転行列はピクチャの投影を決定する 3 行 3 列の行列です。この行列は次式に従って 3 次元座標 (x, y, z) をピクチャ座標 (x', y', z') に変換します。

```
| x' |   | m00 m01 m02 |   | x |   | m00 * x + m01 * y + m02 * z |
| y' | = | m10 m11 m12 | * | y | = | m10 * x + m11 * y + m12 * z |
| z' |   | m20 m21 m22 |   | z |   | m20 * x + m21 * y + m22 * z |
```

このピクチャの回転行列を得ます。
```
const G3Rotation* GetRotation() const;
G3Rotation* GetRotation();
```
戻り値　G3Rotation オブジェクトへのポインタを返します。通常ピクチャでなければ null ポインタを返します。

9.2.4 表示マスク

このオブジェクトの表示マスクを得ます。
```
const AwVisibleMask* GetVisibleMask() const;
AwVisibleMask* GetVisibleMask();
```
戻り値　AwVisibleMask オブジェクトへのポインタを返します。

例）ピクチャ 1 のクラス 5 も表示可にする。
```
AwPicture* pPicture = pModel->GetPictureList()->GetPicture(1);
if (pPicture == NULL)
  return;
pPicture->GetVisibleMask()->SetVisible(AwItemAttributes::BY_CLASS, 5, true);
```

例）アクティブピクチャの表示するクラスを 1 から 100 と 201 から 253 と 255 にする。また製図アイテムと点アイテムを表示しないようにする。

```
AwPicture* pPicture = pModel->GetPictureList()->CreateActivePicture();
```

```
  if (pPicture == NULL)
    return;
  AwVisibleMask* pMask = pPicture->GetVisibleMask();
  if (pMask == NULL)
    return;
  // CLS/DSP REL ALL ADD 1 -100 201 -253 255 <CE>
  pMask->SetVisibleAll(AwItemAttributes::BY_CLASS, false);          // 全クラス
  pMask->SetVisibles(AwItemAttributes::BY_CLASS, 1, 100, true);     // クラス 1-100
  pMask->SetVisibles(AwItemAttributes::BY_CLASS, 201, 253, true);   // クラス 201-253
  pMask->SetVisible(AwItemAttributes::BY_CLASS, 255, true);         // クラス 255
  // ITM/DSP ADD MANY REL MDRF MPNT <CE>
  pMask->SetVisibleAll(AwItemAttributes::BY_ITEMTYPE, true);        // 全タイプ
  pMask->SetVisible(AwItemAttributes::BY_ITEMTYPE, AwItem::POINT, false);
  pMask->SetVisible(AwItemAttributes::BY_ITEMTYPE, AwMaskSet::MASK_DRAFTING, false); // 製図アイテム
```

9.2.5 ウインドウ

ウインドウは通常ピクチャ上に定義した矩形領域です。このウインドウをドローイングレイアウトに配置します。通常ピクチャはデフォルトウインドウを持っています。デフォルトウインドウはウインドウ番号が 0 で、定義しなくても自動的に作成されるウインドウのことです。

このピクチャのウインドウを得ます。

```
const AwWindow* GetWindow(int wid) const;
AwWindow* GetWindow(int wid);
```

wid　　ウインドウ番号（0 -)。

戻り値　ウインドウ番号が wid の AwWindow オブジェクトへのポインタを返します。オブジェクトが存在しないときは null ポインタを返します。

例) ピクチャ＃１、ウインドウ＃０のウインドウ原点を (10,20) にする。

```
  AwPicture* pPicture = pModel->GetPictureList()->GetPicture(1);
  if (pPicture == NULL)
    return;
  AwWindow* pWindow = pPicture->GetWindow(0);
  if (pWindow == NULL)
    return;
  pWindow->SetOrigin(G2Point(10.0, 20.0));
```

例) ピクチャ＃１、ウインドウ＃２のウインドウゾーンを (-10,-20)(100,200) にする。

```
  AwPicture* pPicture = pModel->GetPictureList()->GetPicture(1);
  if (pPicture == NULL)
    return;
  AwWindow* pWindow = pPicture->GetWindow(2);
  if (pWindow == NULL)
    return;
  const double size[] = { 110.0, 220.0 };
  pWindow->SetRect(G2Point(-10.0, -20.0), size, 0.0);
```

9.2.6 Getter

このピクチャのグリッドを得ます。

```
const AwGridSet* GetGridSet() const;
AwGridSet* GetGridSet();
```

戻り値　AwGridSet オブジェクトへのポインタを返します。

このピクチャの補助座標を得ます。

```
const AwWcs& GetWcs() const;
AwWcs* GetWcs();
```

戻り値　AwWcs オブジェクトへのポインタを返します。

9.3 AwVisibleMask クラス

このクラスはアイテムの表示を制御するマスクセットを表現します。このクラスは AwMaskSet クラスの派生クラスです。
AwVisibleMask オブジェクトは AwPicture オブジェクトから得ます。

9.3.1 Getter

指定したアイテムが表示可能か判定します。
```
bool IsVisible(const AwItemAttributes& attr) const;
```
attr　検査する AwItemAttributes オブジェクト。
戻り値　AwItemAttributes オブジェクトのアイテムタイプ、クラス、レビジョン、線種、線幅の全てが表示可能なら true を返します。

指定したアイテムが表示可能か判定します。
```
bool IsVisible(const AwSrAttributes& srAttr) const;
```
srAttr　検査する AwSrAttributes オブジェクト。
戻り値　AwSrAttributes オブジェクトのアイテムタイプ、クラス、レビジョン、線種、線幅の全てが表示可能なら true を返します。

指定したマスクの要素が表示可能か判定します。
```
bool IsVisible(int type, int no) const;
```
type　マスクの種類を示す整数。
no　要素の番号 (1-)。
戻り値　指定したマスクの要素が表示可能なら true を返します。

指定したマスクの全ての要素が表示可能か判定します（デフォルトの状態)。
```
bool IsVisibleAll(int type) const;
```
type　マスクの種類を示す整数。
戻り値　指定したマスクの全ての要素が表示可能なら true を返します。

9.3.2 Setter

指定したマスクのひとつの要素の値を設定します。
```
void SetVisible(int type, int no, bool value);
```
type　マスクの種類を示す整数。
no　設定する要素の番号 (1-)。
value　表示するなら true、表示しないなら false。

指定したマスクの連続する複数の要素の値を設定します。
```
void SetVisibles(int type, int from, int to, bool value);
```
type　マスクの種類を示す整数。
from　設定する最初の要素番号 (1-)。
to　設定する最後の要素番号 (1 <= from <= to)。
value　表示するなら true、表示しないなら false。

指定したマスクの全ての要素の値を設定します。
```
void SetVisibleAll(int type, bool value);
```
type　マスクの種類を示す整数。
value　表示するなら true、表示しないなら false。

9.4 AwWindow クラス

このクラスはピクチャ上に定義した表示範囲 (矩形）を表現します。
ウインドウ番号 が 0 のウインドウ は、定義しなくても自動的に用意されるデフォルトウインドウです。
デフォルトウインドウの表示範囲は指定することができず、常にピクチャの全体となります。デフォルトウインドウは基準点のみ変更できます。
ウインドウ番号は 1 から始まる整数です。
以下の項目はデフォルトウインドウ以外に有効です。

- **矩形の位置（矩形の隅の点。通常、矩形の左下の点）**
- **矩形の大きさ（矩形の横幅と縦幅）**
- **矩形の底辺の角度**
- **名前**

AwWindow オブジェクトは AwPicture オブジェクトから得ます。

9.4.1 コンストラクタ

```
AwWindow(int id);
```
id　　ウインドウ番号。

9.4.2 Getter

このウインドウの番号を得ます。
```
int GetId() const;
```
戻り値　ウインドウ番号を返します。

このオブジェクトが有効なウインドウか判定します。
```
bool IsValid() const;
```
戻り値　このオブジェクトが有効な表示範囲を持っているなら true を返します。

9.4.3 基準点

基準点はこのウインドウをドローイングレイアウトに配置するとき基準になる点です。
このウインドウの基準点を取得／設定します。
```
const G2Point& GetOrigin() const;
void SetOrigin(const G2Point& point);
```

9.4.4 表示範囲

このウインドウの表示範囲を取得／設定します。デフォルトウインドウには表示範囲を設定できません。デフォルトウインドウの表示範囲はピクチャに依存します。

このウインドウの表示範囲を設定します。
```
void SetRect(const G2Point& location, const double size[], double angle);
```
location　矩形の位置。
size　　矩形の大きさ。size[0]: 横幅、size[1]: 縦幅。0 < size[0], size[1]。
angle　　矩形底辺の角度 (単位は度)。

このウインドウの表示範囲を矩形で設定します。角度は 0、位置は矩形の左下隅をとります。
```
void SetRect(const G2Rect& rect);
```
rect　　表示範囲を表す矩形。

このウインドウの表示範囲を得ます。

```
void GetRect(G2Point* location, double size[], double* angle, bool preciseW0 = false) const;
```
location　矩形の位置を格納する G2Point オブジェクトへのポインタ。
size　矩形の大きさを格納する配列。size[0]: 横幅、size[1]: 縦幅。0 < size[0], size[1]。
angle　矩形底辺の角度 (単位は度)。
preciseW0　デフォルト・ウインドウの表示範囲を計算するときの選択肢。
false　大まかな大きさを計算する。
true　精密な大きさを計算する。

このウインドウの角度を得ます。
```
double GetAngle() const;
```
戻り値　ウインドウの角度 (単位は度) を返します。

このウインドウの 4 隅の点を計算します。
```
void GetBoundPoints(G2Point points[], bool preciseW0 = false, double clearance = 0.0) const;
```
points　表示範囲の 4 隅の点を格納する配列。点の順序は（左下、右下、右上、左上) です。
preciseW0　デフォルト・ウインドウの表示範囲を計算するときの選択肢。
false　大まかな大きさを計算する。
true　精密な大きさを計算する。
clearance　クリアランス。clearance > 0 なら拡大した矩形の 4 隅の点を計算します。clearance < 0 なら縮小した矩形の 4 隅の点を計算します。縮小した矩形がゼロまたは負になるようならこの引数は無視します。

9.4.5　名前

このウインドウの名前を取得／設定します。同一のピクチャに属すウインドウの名前は重複してはなりません。デフォルトウインドウには名前をつけることはできません。

このウインドウの名前を得ます。
```
const std::string& GetName() const;
```

このウインドウに名前をつけます。このメソッドは名前の重複検査は行いません。
```
void  SetName(const std::string& name);
```

このウインドウの名前を除去します。
```
void RemoveName();
```

9.5　AwGridSet クラス

このクラスはピクチャ上に定義したグリッドセットを表現します。
AwGridSet オブジェクトは AwPicture オブジェクトから得ます。

グリッド点を計算します。
```
G2Point CalcPoint(const G2Point& point) const;
```
point　入力点。
戻り値　入力点に最も近いグリッド点を返します。「グリッドを適用しない状態」なら入力点をそのまま返します。

9.6　AwWcs クラス

このクラスはピクチャ上に定義した補助座標系を表現します。補助座標系は適用する／適用しないの状態を持ちます。補助座標系の変換行列を設定 / 変更しても、適用しないの状態では設定値は有効にはなりません。補助座標系を適用する状態にしたときに変換行列が有効になります。
AwWcs オブジェクトは AwPicture オブジェクトから得ます。

この補助座標の状態を取得／設定します。
```
bool IsActive() const;
```
　戻り値　true：この補助座標を適用する、false：この補助座標を適用しない。
```
void SetActive(bool active);
```
　active　true：この補助座標を適用する、false：この補助座標を適用しない。

この補助座標の座標変換行列を取得します。
```
const G2Matrix& GetMatrix() const;
```
　戻り値　G2Matrix オブジェクトの参照を返します。この補助座標を適用しないの時の G2Matrix オブジェクトは恒等変換行列です。

この補助座標の回転を取得します。this->GetMatrix().GetVector() の簡略形です。
```
G2Vector GetRotation() const;
```
　戻り値　座標変換行列の回転成分の X 軸方向余弦を返します

9.7　G3Rotation クラス

このクラスは 3 次元座標の回転行列 (3x3) を表現します。この行列は直交行列です。座標変換は次のようになります。
```
| x' |   | m00 m01 m02 |   | x |   | m00 * x + m01 * y + m02 * z |
| y' | = | m10 m11 m12 | * | y | = | m10 * x + m11 * y + m12 * z |
| z' |   | m20 m21 m22 |   | z |   | m20 * x + m21 * y + m22 * z |
```

これは、ピクチャの座標系を定義するときにも使用します。
　列 (m00, m10, m20) -> X 軸方向余弦 (x,y,z)
　列 (m01, m11, m21) -> Y 軸方向余弦 (x,y,z)
　列 (m02, m12, m22) -> Z 軸方向余弦 (x,y,z)

アイソメトリック・ヴュー (投影の三つの軸の長さ比が同じ）の回転行列は次のように計算できます。尚、回転角は座標軸の正の方向から原点を見て反時計回りを正とします。
```
G3Rotation m1, m2;
m1.SetRotation(3, 0.25 * G2Math::Pi); // 最初に Z 軸 45 度回転する
m2.SetRotation(1, -atan(sqrt(2.0)));  // 次に X 軸 -54.74 度回転する
G3Rotation iso(m2 * m1);              // ふたつの回転を連結する
```

座標変換を行うメソッドは Transform() です。逆変換を行うには逆行列を使って行いますが、簡便のためのメソッド InverseTransform() があります。次の例では、論理的には p2 == point ですが、数値計算上の誤差があり p2 != point となることがあります。
```
G3Point point(1.0, 2.0, 3.0);
G3Point p1, p2;
iso.Transform(point, &p1);
iso.InverseTransform(p1, &p2);
```

9.7.1　コンストラクタ

```
G3Rotation();
```
　単位行列になります。

9.7.2　Getter

この行列オブジェクトの妥当性を検査します。
```
bool IsValid() const;
```
　戻り値　true：この行列オブジェクトが 3x3 直交行列なら true を返します。

この行列オブジェクトの方向余弦を得ます。
```
G3Point GetAxis(int axis) const;
```
axis　　座標軸を指定する整数 (1=X、2=Y、3=Z)。
戻り値　この行列オブジェクトの座標軸 axis の方向余弦を返します。

この行列オブジェクトの X 軸の方向余弦を得ます。
```
G3Point GetXaxis() const;
```
戻り値　この行列オブジェクトの X 軸方向余弦を返します。
この行列オブジェクトの Y 軸の方向余弦を得ます。
```
G3Point GetYaxis() const;
```
戻り値　この行列オブジェクトの Y 軸方向余弦を返します。
この行列オブジェクトの Z 軸の方向余弦を得ます。
```
G3Point GetZaxis() const;
```
戻り値　この行列オブジェクトの Z 軸方向余弦を返します。

この行列オブジェクトの転置行列の方向余弦を得ます。これは CreateTranspose().GetAxis(axis) の簡略形です。
```
G3Point GetTransposedAxis(int axis) const;
```
axis　　座標軸を指定する整数 (1=X、2=Y、3=Z)。
戻り値　転置行列の座標軸 axis の方向余弦を返します。

転置行列の X 軸の方向余弦を得ます。
```
G3Point GetTransposedXaxis() const;
```
戻り値　転置行列の X 軸方向余弦を返します。
転置行列の Y 軸の方向余弦を得ます。
```
G3Point GetTransposedYaxis() const;
```
戻り値　転置行列の Y 軸方向余弦を返します。
転置行列の Z 軸の方向余弦を得ます。
```
G3Point GetTransposedZaxis() const;
```
戻り値　転置行列の Z 軸方向余弦を返します。

9.7.3 Setter

この行列オブジェクトを単位行列に設定します。
```
void SetIdentity();
```

この行列オブジェクトに回転軸、回転角を指定して回転を設定します。
```
void SetRotation(int axis, double theta);
```
axis　　座標軸を指定する整数 (1=X、2=Y、3=Z)。
theta　　回転角 (ラジアン)。

X 軸回りの回転行列

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & \mathrm{COS(ANG)} & -\mathrm{SIN(ANG)} \\ 0 & \mathrm{SIN(ANG)} & \mathrm{COS(ANG)} \end{bmatrix}$$

Y 軸回りの回転行列

$$\begin{bmatrix} \mathrm{COS(ANG)} & 0 & \mathrm{SIN(ANG)} \\ 0 & 1 & 0 \\ -\mathrm{SIN(ANG)} & 0 & \mathrm{COS(ANG)} \end{bmatrix}$$

Z 軸回りの回転行列

$$\begin{bmatrix} COS(ANG) & -SIN(ANG) & 0 \\ SIN(ANG) & COS(ANG) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

この行列オブジェクトに、投影法に従ったピクチャの回転行列を設定します。

```
void SetStandardRotation(int type, int mtxid);
```

type　投影法を指定する整数 (0、1)。

mtxid　ヴューの番号 (1-7)。

投影タイプが 0 のとき

1 : 平面 (no rotation)
2 : 正面 (rotate X-90)
3 : 右側面 (rotate Y-90)
4 : 底面 (rotate X180)
5 : 背面 (rotate X90)
6 : 左側面 (rotate Y90)
7 : アイソメトリック (rotate Z45, X-54.74)

投影タイプが 1 のとき

1 : 平面 (no rotation)
2 : 正面 (rotate X-90)
3 : 右側面 (rotate X-90, Y-90)
4 : 底面 (rotate X180)
5 : 背面 (rotate X-90, Y180)
6 : 左側面 (rotate X-90, Y90)
7 : アイソメトリック (rotate Z45, X-54.74)

9.7.4 座標変換

点を座標変換します。

```
void Transform(G3Point* point) const;
```

point　入力点かつ出力点。

点を座標変換します。

```
void Transform(const G3Point& src, G3Point* dst) const;
```

src　入力点。

dst　出力点。

点列を座標変換します。

```
void Transform(const G3Point src[], int srcOff, G3Point dst[], int dstOff, int count) const;
```

src　入力点列。

srcOff　入力点列の開始インデックス (0 -)。

dst　出力点列。

dstOff　出力点列の開始インデックス (0 -)。

count　変換する点数。

src[srcOff] から src[srcOff + count - 1] までを座標変換し、dst[dstOff] から dst[dstOff + count - 1] に格納します。

点列を座標変換します。

```
void Transform(const double src[], int srcOff, double dst[], int dstOff, int count) const;
```

src　入力点列。3 次元座標（x0,y0,z0,x1,y1,z1, ...）を持つ double 配列。

srcOff　入力点列の開始インデックス (0 -)。

dst　出力点列。3 次元座標（x0,y0,z0,x1,y1,z1, ...）を格納する double 配列。

dstOff　出力点列の開始インデックス (0 -)。

count　　変換する点数。
src[srcOff] から src[srcOff + 3*count - 1] までを座標変換し、dst[dstOff] から dst[dstOff + 3*count - 1] に格納します。

9.7.5　逆座標変換

点を逆座標変換します。
```
void InverseTransform(G3Point* point) const;
```
point　　入力点かつ出力点。

点を逆座標変換します。
```
void InverseTransform(const G3Point& src, G3Point* dst) const;
```
src　　入力点。
dst　　出力点。

点列を逆座標変換します。
```
void InverseTransform(const G3Point src[], int srcOff, G3Point dst[], int dstOff, int count) const;
```
src　　入力点列。
srcOff　　入力点列の開始インデックス (0 -)。
dst　　出力点列。
dstOff　　出力点列の開始インデックス (0 -)。
count　　変換する点数。
src[srcOff] から src[srcOff + count - 1] までを座標変換し、dst[dstOff] から dst[dstOff + count - 1] に格納します。

点列を逆座標変換します。
```
void InverseTransform(const double src[], int srcOff, double dst[], int dstOff, int count) const;
```
src　　入力点列。3 次元座標（x0,y0,z0,x1,y1,z1, ...）を持つ double 配列。
srcOff　　入力点列の開始インデックス (0 -)。
dst　　出力点列。3 次元座標（x0,y0,z0,x1,y1,z1, ...）を格納する double 配列。
dstOff　　出力点列の開始インデックス (0 -)。
count　　変換する点数。
src[srcOff] から src[srcOff + 3*count - 1] までを座標変換し、dst[dstOff] から dst[dstOff + 3*count - 1] に格納します。

9.7.6　演算子

行列の積を計算する * 演算子と =* 演算子 があります。* 演算子は行列の積を新しい G3Rotation オブジェクトで返します。これに対して =* 演算子 は演算子を呼び出したオブジェクトを計算結果で書き換えます。

この行列オブジェクトに引数の行列を連結した行列オブジェクトを計算します (this * rotation)。
```
G3Rotation operator*(const G3Rotation& rotation) const;
```
rotaqtion　　G3Rotation オブジェクトの参照。
戻り値　　この行列オブジェクトに引数の行列を連結した行列オブジェクトを返します。

この行列オブジェクトに引数の行列を連結します。(this = this * rotation)。
```
G3Rotation& operator*=(const G3Rotation& rotation);
```
rotaqtion　　G3Rotation オブジェクトの参照。
戻り値　　この行列オブジェクトの参照を返します。

9.7.7 転置行列

転置行列とは行列の (i, j) 要素と (j, i) 要素を交換した行列です。このオブジェクトが直交行列であれば転置行列は逆行列と同じです。

```
      元の行列          転置行列
   | m00 m01 m02 |   | m00 m10 m20 |
   | m10 m11 m12 |   | m01 m11 m21 |
   | m20 m21 m22 |   | m02 m12 m22 |
```

この行列オブジェクトの転置行列を得ます。

```
G3Rotation CreateTranspose() const;
```

戻り値　　この行列オブジェクトの転置行列オブジェクトを返します。

9.8 G3Point クラス

このクラスは 3 次元空間の点またはベクトルを表現します。

9.8.1 コンストラクタ

```
G3Point();
```

座標 (0, 0, 0) の点オブジェクト。

```
G3Point(double x, double y, double z);
```

座標 (x, y, z) の点オブジェクト。

9.8.2 Setter

このオブジェクトに座標 (x, y, z) を設定します。

```
void Set(double x, double y, double z);
```

9.8.3 演算子

点の和、差、スカラー積、スカラー除を計算する演算子 (+、-、*、/) があります。これらの演算子は計算結果を新しい G3Point オブジェクトで返します。これに対応する代入を伴う演算子 (+=、-=、*=、/=) は演算子を呼び出したオブジェクトを計算結果で書き換えます。

点座標の和 (this + point) を計算します。

```
G3Point operator+(const G3Point& point) const;
```

point　　加える点。
戻り値　　点座標の和のオブジェクトを返します。

点座標の加算。このオブジェクトに点 point を加えます (this = this + point)。

```
G3Point& operator+=(const G3Point& point);
```

point　　加える点。
戻り値　　このオブジェクトの参照を返します。

点座標の差 (this - point) を計算します。

```
G3Point operator-(const G3Point& point) const;
```

point　　引く点。
戻り値　　点座標の差のオブジェクトを返します。

点座標の減算。このオブジェクトから点 point を引きます (this = this - point)。

```
G3Point& operator-=(const G3Point& point);
```

point　　引く点。

戻り値　　このオブジェクトの参照を返します。

点座標のスカラー積 (this * s) を計算します。
```
G3Point operator*(double s) const;
```
s　　乗数。
戻り値　　点座標のスカラー積のオブジェクトを返します。

点座標のスカラー積。このオブジェクトの座標にスカラー s を掛けます (this = this * s)。
```
G3Point& operator*=(double s);
```
s　　乗数。
戻り値　　このオブジェクトの参照を返します。

点座標のスカラー除 (this / s) を計算します。
```
G3Point operator/(double s) const;
```
s　　除数。s != 0.0。
戻り値　　この点座標のスカラー除のオブジェクトを返します。

点座標のスカラー除。このオブジェクトの座標をスカラー s で割ります (this = this / s)。
```
G3Point& operator/=(double s);
```
s　　除数。s != 0.0。
戻り値　　このオブジェクトの参照を返します。

以下に簡単な例を示します。
```
G3Point p1( 5.0,  4.0,  3.0);
G3Point p2( 2.0,  3.0,  4.0);
G3Point p3;
p3 = p1 + p2;    // p3 (7.0, 7.0, 7.0)
p3 = p1 - p2;    // p3 (3.0, 1.0, -1.0)
p3 = p1 * 0.5;   // p3 (2.5, 2.0, 1.5)
p3 = 0.5 * p1;   // コンパイルエラーになります。
p3 = p1 / 2.0;   // p3 (2.5, 2.0, 1.5)
p1 += p2;        // p1 (7.0, 7.0, 7.0)
p1 -= p2;        // p1 (5.0, 4.0, 3.0)
p3 = (p1 + p2) / 2.0;    // p3 (3.5, 3.5, 3.5) = p1 と p2 の中点。
```

9.8.4　二点間距離

２点の距離を計算します。
```
double CalcDistance(const G3Point& point) const;
```

２点の距離の自乗を計算します。
```
double CalcSquareDistance(const G3Point& point) const;
```

２点の L1 距離を計算します。L1 距離は 2 点の座標の差分 dx, dy, dz の絶対値の和です。
L1 = | this.x - point.x | + | this.y - point.y | + | this.z - point.z |
```
double CalcL1Distance(const G3Point& point) const;
```

２点の L ∞距離を計算します。
L ∞距離は 2 点の座標の差分 dx, dy, dz の絶対値の最も大きいものです。
L ∞ = Max(| this.x - point.x |, | this.y - point.y |, | this.z - point.z |)。
```
double CalcLinfDistance(const G3Point& point) const;
```

9.8.5　ベクトル

3D ベクトルを表現するクラスはありませんので、G3Point クラスを使います。G3Point は 3D ベクトルに関係するメソッドを持ちます。

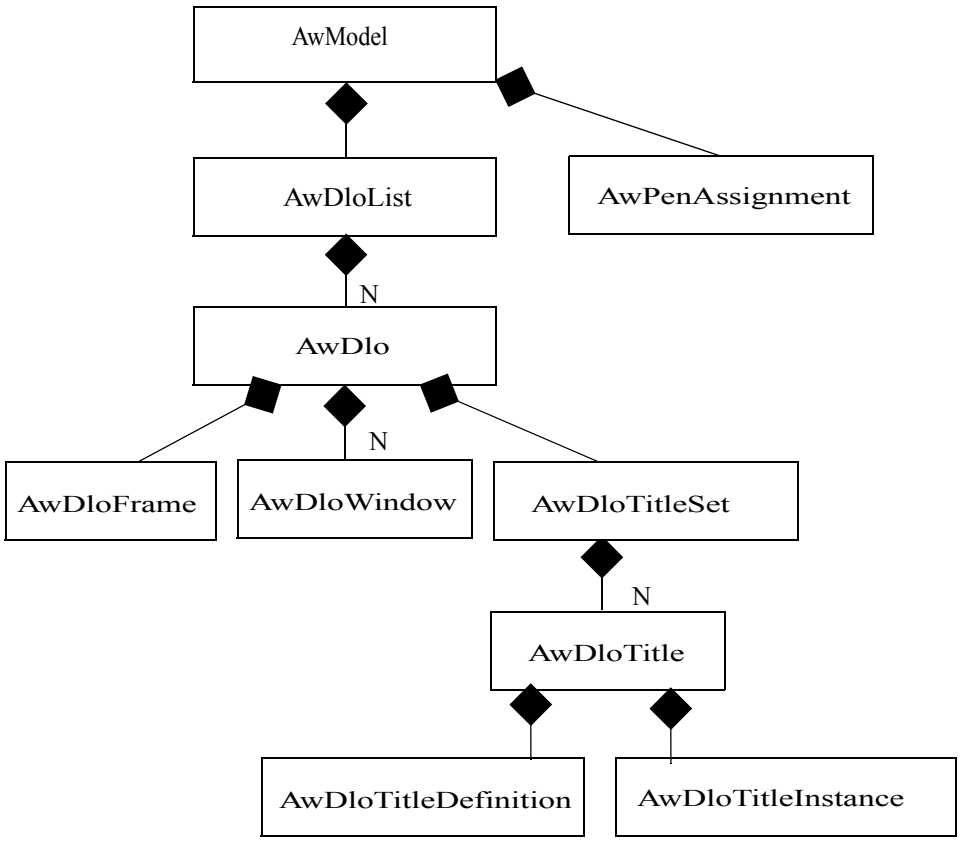
ベクトルの内積を計算します。
```
double Dot(const G3Point& vector) const;
```
vector　　ベクトル。
戻り値　　このベクトルと引数のベクトルの内積を返します。

ベクトルの外積を計算します。
```
G3Point Cross(const G3Point& vector) const;
```
vector　　ベクトル。
戻り値　　このベクトルと引数のベクトルの外積を返します。

このベクトルを単位ベクトルにします。
```
double Normalize();
```
戻り値　　元のベクトルの長さを返します。

第 10 章　ドローイング

この章ではアプリケーションモデル AwModel オブジェクトが管理するドローイングレイアウトを説明します。下記はこの章で解説する主なクラスを含むクラス図です。



10.1 AwDloList クラス

このクラスは AwDlo オブジェクトのコンテナです。ドローイングレイアウトの共通情報を持ち、AwDlo オブジェクトを管理 (所有）します。
AwDloList オブジェクトは AwModel オブジェクトから得ます。

10.1.1 ドローイングスケール

ドローイングスケール が図面縮尺、ピクチャスケールが部分拡大図倍率に相当します。ドローイングスケールはひとつだけで、ずべてのドローイングレイアウトに共通です。

文字やマークのピクチャ上の大きさはピクチャスケールとドローイングスケールを反映したものです。通常、ピクチャスケールとドローイングスケールはアイテム作成前に設定します。アイテム作成後にスケールを変更しても、それは文字やマークの大きさには反映しません。これを避けるにはアイテム作成後のスケールの変更は行わないことです。

ドローイングスケールを得ます。
```
double GetDrawingScale() const;
```
戻り値　ドローイングスケールを返します。

ドローイングスケールを設定します。
```
bool SetDrawingScale(double scale);
```
scale　ドローイングスケール。1.0e-8 <= scale。

10.1.2 ドローイングモード

アクティブなドローイングレイアウト番号を得ます。現在のドローイングレイアウト番号、または最近ドローイングモードに切り替えたときに使われたドローイングレイアウト番号です。一度もドローイングモードにしたことがなければ 0 です。
```
int GetActiveDlold() const;
```
戻り値　ドローイングレイアウト番号を返します。

ドローイングモードか判定します。
```
bool IsDrawingMode();
```
戻り値　ドローイングレイアウトモードのとき true を返します。

10.1.3 ドローイングレイアウトを得る

指定した番号のドローイングレイアウトを得ます。
```
const AwDlo* GetDlo(int did) const;
AwDlo* GetDlo(int did);
```
did　ドローイングレイアウト番号。1 <= did。
戻り値　ドローイングレイアウト番号が did の AwDlo オブジェクトへのポインタを返します。オブジェクトが存在しないときは null ポインタを返します。

指定した名前のドローイングレイアウトを得ます。
```
const AwDlo* GetDlo(const std::string& name) const;
```
name　ドローイングレイアウトの名前。
戻り値　レイアウト名が name の AwDlo オブジェクトへのポインタを返します。オブジェクトが存在しないときは null ポインタを返します。

指定した番号のドローイングレイアウトを作成します。それが既にあるときは作成しません。
```
AwDlo* CreateDlo(int did);
```
did　ドローイングレイアウト番号。1 <= did。
戻り値　ドローイングレイアウト番号が did の AwDlo オブジェクトへのポインタを作成します。ドローイングレイアウト番号の誤りなどでオブジェクトが作成できないときは null ポインタを返します。

10.2 AwDlo クラス

このクラスはドローイングレイアウトを表現します。
ドローイングレイアウトは下記の情報を持っています。

- **名前**（デフォルト値は空文字列）
- **図枠**（デフォルトの状態は図枠未指定）
- **ドローイングタイトル**（デフォルトの状態は空）
- **ウインドウ配置**（デフォルトの状態は空）

AwDlo オブジェクトは AwDloList オブジェクトから得ます。

10.2.1 クラスメソッド

ドローイングレイアウトの番号を判定します。ドローイングレイアウト番号は 1 から始まる整数です（1 ?AwDlo::MAX_DLO_ID）。

```
static bool IsValidId(int id);
```

id　　判定する番号。
戻り値　id がドローイングレイアウトの番号として適切な値であるとき true を返します。

10.2.2 Getter

このドローイングレイアウトの番号を得ます。

```
int GetId() const;
```

戻り値　このドローイングレイアウトの番号を返します。

このドローイングレイアウトの図枠を得ます。

```
const AwDloFrame* GetDloFrame() const;
AwDloFrame* GetDloFrame();
```

戻り値　このドローイングレイアウトの図枠 AwDloFrame オブジェクトへのポインタを返します。オブジェクトが存在しないときは null ポインタを返します。

このドローイングレイアウトのタイトルセットを得ます。

```
const AwDloTitleSet* GetDloTitleSet() const;
AwDloTitleSet* GetDloTitleSet();
```

戻り値　このドローイングレイアウトのタイトルセット AwDloTitleSet オブジェクトへのポインタを返します。オブジェクトが存在しないときは null ポインタを返します。

10.2.3 名前

このドローイングレイアウトの名前を取得／設定します。名前は重複してはなりません。

このドローイングレイアウトの名前を得ます。

```
const std::string& GetName() const;
```

このドローイングレイアウトに名前をつけます。このメソッドは名前の重複検査は行いません。

```
void  SetName(const std::string& name);
```

このドローイングレイアウトの名前を除去します。

```
void RemoveName();
```

10.2.4 ウインドウ配置

ドローイングレイアウトのウインドウ配置数の上限は、個々のドローイングレイアウト毎ではなく、全てのドローイングレイアウトのウインドウ配置数の合計で制限しています。

このドローイングレイアウトに配置されたウインドウ数を得ます。
```
int GetDloWindowCount() const;
```
戻り値　このドローイングレイアウトのウインドウ数を返します。

このドローイングレイアウトにウインドウを配置します。同じウインドウを複数配置することはできません。
```
bool AddDloWindow(const AwDloWindow& window);
```
window　ウインドウ配置。
戻り値　ウインドウ配置 window が、このドローイングレイアウトになければ、これを追加します。既にあれば、それを window の値で更新します。ウインドウ配置を追加したか更新したら true を返します。

このドローイングレイアウトから指定したウインドウ配置を削除します。
```
bool RemoveDloWindow(int pid, int wid);
```
pid　ピクチャ番号。
wid　ウインドウ番号。
戻り値　指定したウインドウ配置を削除したら true を返します。

このドローイングレイアウトから指定したピクチャを参照するウインドウ配置をずべて削除します。
```
bool RemoveDloWindows(int pid);
```
pid　ピクチャ番号。
戻り値　ウインドウ配置を削除したら true を返します。

このドローイングレイアウトの全てのウインドウ配置を削除します。
```
void RemoveAllDloWindows();
```

10.3 AwDloWindow クラス

このクラスはドローイングレイアウトに配置したウィンドウを表現します。
ウィンドウ配置は下記の情報を持っています。
- 参照するウインドウ（ピクチャ番号とウインドウ番号で特定する）
- 配置位置（ドローイングレイアウト座標）
- 回転角度（デフォルト値は 0 度）
- 倍率（デフォルト値は 1.0）

配置するウインドウはあらかじめピクチャに定義されているものとします。このクラスのメソッドはピクチャ番号とウインドウ番号の値が適切な値であるかどうかは検査しますが、ピクチャ番号とウインドウ番号で特定するウインドウが定義されているかどうかは検査しません。

10.3.1 コンストラクタ

参照するウインドウを指定するコンストラクタ。
```
AwWindow(AwDloWindow(int pid, int wid));
```
pid　ピクチャ番号。
wid　ウインドウ番号。

10.3.2 Getter

このオブジェクトが参照するウインドウのピクチャ番号を得ます。
int GetPictureId() const;
　戻り値　ピクチャ番号を返します。

このオブジェクトが参照するウインドウのウインドウ番号を得ます。
int GetWindowId() const;
　戻り値　ウインドウ番号を返します。

このオブジェクトが有効な値を持つか判定します。
bool IsValid() const;
　戻り値　このオブジェクトが有効な値を持っているなら true を返します。

10.3.3 配置パラメータ

このウインドウの配置点を取得／設定します。配置点はドローイングレイアウト座標です。
const G2Point& GetOrigin() const;
　このウインドウの配置点を返します。
void SetOrigin(const G2Point& point);
　このウインドウの配置点を設定します。

このウインドウの回転角を取得／設定します。
double GetAngle() const;
　このウインドウの回転角を返します。
void SetAngle(double angle);
　このウインドウの回転角 (angle 0-360 度）を設定します。

このウインドウの倍率を取得／設定します。
double GetScale() const;
　このウインドウの倍率を返します。
void SetScale(double scale);
　このウインドウの倍率 (0.0 < scale）を設定します。

10.4 AwDloFrame クラス

このクラスはドローイングレイアウトの図枠を表現します。
AwDloFrame オブジェクトは AwDlo オブジェクトから得ます。このオブジェクトの状態が「無効」であるときは、AwDlo オブジェクトは初期化されただけで、適切な図枠が設定されていない状態であるとみなされます。そのような状態の AwDlo オブジェクトを指定してドローイングモードに切り替えることはできません。

10.4.1 定数

図枠の種類

定数名	説明
AwDloFrame::TYPE_NONE	不定
AwDloFrame::TYPE_SYMBOL	図枠シンボル
AwDloFrame::TYPE_FREESIZE	フリーサイズ
AwDloFrame::TYPE_STANDARD	規格サイズ

規格サイズ

定数名	説明
AwDloFrame::SIZE_A0	841 x 1189 mm
AwDloFrame::SIZE_A1	594 x 841 mm
AwDloFrame::SIZE_A2	420 x 594 mm
AwDloFrame::SIZE_A3	297 x 420 mm
AwDloFrame::SIZE_A4	210 x 297 mm
AwDloFrame::LANDSCAPE	ランドスケープ（横長）
AwDloFrame::PORTRAIT	ポートレイト（縦長）

10.4.2 Getter

このオブジェクトが有効であるか判定します。「有効」とは、このオブジェクトが、適切な図枠種類、とサイズを持っていることです。
```
bool IsValid() const;
```
戻り値　このオブジェクトが有効と判断したら true を返します。

このオブジェクトの図枠種類を得ます。
```
int GetType() const;
```
戻り値　このオブジェクトの図枠種類を返します。AwDloFrame::TYPE_NONE は図枠不定です。

このオブジェクトの図枠サイズを得ます。
```
const G2Rect* GetBounds() const;
```
戻り値　IsValid() が true のとき G2Rect オブジェクトへのポインタを返します。それ以外では null ポインタを返します。

このオブジェクトの図枠サイズを得ます。
```
bool GetSize(int* width, int* height) const;
```
width　図枠の横幅を設定します。
height　図枠の縦幅を設定します。
戻り値　IsValid() が true のとき、width、height を設定し true を返します。それ以外では false を返します。

10.4.3 図枠シンボル

このオブジェクトの図枠シンボル名を得ます（GetType() が AwDloFrame::TYPE_SYMBOL を返すとき）。
```
const std::string& GetSymbolName() const;
```
戻り値　図枠シンボル名を返します。

このオブジェクトにシンボルの図枠を設定します。
```
bool SetSymbolName(const std::string& name);
```
name　図枠シンボル名。ディレクトリやファイル拡張子の無い名前。
戻り値　設定したら true を返します。

このオブジェクトにシンボルをロードします。
```
bool LoadSymbol();
```
戻り値　成功したら true を返します。

SetSymbolName() メソッドは、シンボル名だけを設定し、図枠サイズを不定にします。図枠サイズを確定させるには LoadSymbol() メソッドを呼び出さなくてはなりません。LoadSymbol() メソッドはシンボルファイルを読み込み、図枠サイズを確定します。同時にタイトル定義を集めてドローングタイトルの設定も行います。

10.4.4 フリーサイズ

このオブジェクトにフリーサイズの図枠を設定します。
```
bool SetFreeSize(int width, int height);
```
width　　横幅 (0 < width <= 32767)。
height　　縦幅 (0 < height <= 32767)。
戻り値　　設定したら true を返します。

10.4.5 規格サイズ

このオブジェクトの規格サイズを得ます（GetType() が AwDloFrame::TYPE_STANDARD を返すとき）。
```
int GetStandardSize() const;
```
戻り値　　規格サイズを表す整数を返します。
```
int GetOrientation() const;
```
戻り値　　図面の向きを表す整数を返します。

このオブジェクトに規格サイズの図枠を設定します。
```
bool SetStandardSize(int size, int orientation);
```
size　　規格サイズを表す整数。
orientation　図面の向きを表す整数。
戻り値　　設定したら true を返します。

10.5 AwDloTitleSet クラス

このクラスはドローイングレイアウトのタイトル全般の情報を持ち、ドローイングレイアウトのタイトル AwDloTitle オブジェクトを管理（所有）します。
ドローイングレイアウトのタイトルの定義は図枠シンボルに埋め込まれていますので、図枠がシンボルの場合のみ有効と考えてください。図枠シンボルを読み込むとき、AwDloTitleSet オブジェクトを設定します。タイトルオブジェクトはタイトルの識別番号で昇順に並べます。

AwDloTitleSet オブジェクトは AwDlo オブジェクトから得ます。

10.5.1 タイトルを得る

このオブジェクトが持つタイトル数を得ます。
```
int GetTitleCount() const;
```
戻り値　　このオブジェクトが持つタイトル数を返します。

タイトルを得ます。
```
const AwDloTitle* GetTitleAt(int index) const
AwDloTitle* GetTitleAt(int index);
```
index　　インデックス。0 <= index < GetTitleCoun()。
戻り値　　指定のインデックスにある AwDloTitle オブジェクトへのポインタを返します。不正なインデックスが与えられると null ポインタを返します。

指定した番号のタイトルを得ます。
```
const AwDloTitle* GetTitle(int tid) const
AwDloTitle* GetTitle(int tid);
```

tid　　タイトル番号。
戻り値　　指定のタイトル番号の AwDloTitle オブジェクトへのポインタを返します。オブジェクトが存在しないときは null ポインタを返します。

10.6 AwDloTitle クラス

このクラスはドローイングレイアウトのタイトルを表現します。このオブジェクトは、タイトルの定義を表現する AwDloTitleDefinition オブジェクトとタイトルの値を持つ AwDloTitleInstance オブジェクトを持ちます。
タイトルの値を持つ AwDloTitleInstance オブジェクトはタイトルの値を持つ他に、表示の制御情報を持ちます。タイトルを非表示にする、表示位置を移動する、文字の大きさを変えるなどです。これらは変更しなければタイトル定義の値を使います。表示の制御情報をクリアすることでタイトル定義に戻すことが出来ます。

図枠シンボル読み込み時にタイトル定義 AwDloTitleDefinition オブジェクトを設定します。このとき、タイトルの値を持つ AwDloTitleInstance オブジェクトは変更しません。図枠シンボルのタイトル定義を変更したり、図枠シンボルが存在しないなどの理由で、タイトル定義がないタイトルが存在することがあります。このようなタイトルは削除はしませんが処理対象ではありません。

AwDloTitle オブジェクトは AwDloTitleSet オブジェクトから得ます。

10.6.1 Getter

このオブジェクトのタイトル番号を得ます。
```
int GetId() const;
```
戻り値　　タイトル番号 を返します。

このタイトルの値を得ます。このオブジェクトがタイトル定義持つ場合に有効なメソッドです。
```
bool GetValue(AwDloTitleInstance* pInstance) const;
```
pInstance　　タイトルの値を得ます。pInstance->IsAutoValue() が true であればその値はタイトル定義から導出したものです。
戻り値　　このオブジェクトがタイトル定義を持たなければ false を返します。タイトルの値を pInstance に設定します。

このタイトルの表示位置を得ます。このオブジェクトがタイトル定義持つ場合に有効なメソッドです。
```
bool GetTextLocation(G2Point* location) const;
```
location　　タイトルの位置を得ます。表示位置が変更されていれば変更値を得ます。
戻り値　　このオブジェクトがタイトル定義を持たなければ false を返します。

このタイトルの文字の大きさを得ます。このオブジェクトがタイトル定義持つ場合に有効なメソッドです。
```
bool GetCharHeight(double* height) const;
```
height　　文字の大きさを得ます。文字の大きさが変更されていれば変更値を得ます。
戻り値　　このオブジェクトがタイトル定義を持たなければ false を返します。

10.6.2 タイトル定義

このオブジェクトのタイトル定義を得ます。
```
const AwDloTitleDefinition* GetDefinition() const;
```
戻り値　　AwDloTitleDefinition オブジェクトへのポインタを返します。オブジェクトが存在しなければ null ポインタを返します。

10.6.3 タイトルインスタンス

このオブジェクトのタイトルインスタンスを得ます。
```
const AwDloTitleInstance* GetInstance() const;
AwDloTitleInstance* GetInstance();
```
戻り値　AwDloTitleInstance オブジェクトへのポインタを返します。オブジェクトが存在しなければ null ポインタを返します。

このオブジェクトにタイトルインスタンスを設定します。このオブジェクトがタイトルインスタンスを持たなければ、タイトルインスタンスを追加します。既に持っていれば、それを更新します。
```
void SetInstance(const AwDloTitleInstance& instance);
```
instance　AwDloTitleInstance オブジェクト。

このオブジェクトのタイトルインスタンスを削除します。
```
bool RemoveInstance();
```
戻り値　このオブジェクトがタイトルインスタンスを持てばそれを削除します。削除したとき true 返します。

10.7 AwDloTitleInstance クラス

10.7.1 コンストラクタ

```
AwDloTitleInstance();
```
コンストラクタ。

10.7.2 文字の大きさ

このタイトルを表示するときの文字の大きさを変更するには以下のメソッドを使います。文字の大きさを変更し、さらに変更フラッグを設定します。SetHeight() メソッドはこのフラッグは変更しません。

このタイトルの文字の大きさを取得／設定します。文字の大きさはドローイングレイアウトスペースで、0.01 から 327.76 の範囲です。
```
double GetHeight() const;
void SetHeight(double height);
```

このタイトルの文字の大きさ変更フラッグを取得／設定します。
```
bool IsHeightModified() const;
void SetHeightModified(bool modified);
```
modified　true : 変更値を使う、false : タイトル定義を参照する。

10.7.3 表示位置

このタイトルを表示する位置を変更するには以下のメソッドを使います。表示位置を変更し、さらに表示位置変更フラッグを設定します。SetLocation() メソッドはこのフラッグは変更しません。

このタイトルの表示位置を取得／設定します。位置の座標はドローイングレイアウトスペースです。
```
const G2Point& GetLocation() const;
void SetLocation(const G2Point& location);
```

このタイトルの表示位置変更フラッグを得ます。
```
bool IsLocationModified() const;
```
このタイトルの表示位置変更フラッグを設定します。
```
void SetLocationModified(bool modified);
```
modified　true : 変更値を使う、false : タイトル定義を参照する。

10.7.4 表示・非表示

このタイトルを非表示にするには以下のメソッドを使います。

このタイトルの表示フラッグを得ます。
```
bool IsVisible() const;
```
このタイトルの表示フラッグを設定します。
```
void SetVisible(bool show);
```
show　　true : 表示、false : 非表示。

10.7.5 タイトルの値

このタイトル値を設定するには以下のメソッドを使います。
タイトル定義が日付けなどのマークアップ、モデルタイトルの引用などで構成されていれば、タイトルの値は自動的に決まりますので、通常はタイトル値の設定はしません。

このタイトルの値を得ます。
```
const std::string& GetText() const;
```
戻り値　　このタイトルの値を返します。

このタイトルの値を設定します。
```
void SetText(const std::string& text);
```
text　　タイトルの値。空のときは値を空にします。

このタイトルの値がタイトル定義から導出されたものか調べます。
```
bool IsAutoValue() const;
```
戻り値　　このタイトルの値がタイトル定義から導出されたものであれば true を返します。
GetValue() メソッドが設定します。

10.8 AwPenAssignment クラス

このクラス はプリンタ／プロッタで使用するペン割り付け表を表現するクラスです。ペン割付けタイプは下記に示すもので、このなかからひとつを選択します。ペン割り付けタイプをアイテムタイプとしていれば、アイテムタイプのペン割り付け表を参照してペン番号を決めます。

AwItemAttributes::BY_ITEMTYPE	アイテムタイプ
AwItemAttributes::BY_CLASS	アイテムクラス
AwItemAttributes::BY_REVISION	アイテムレビジョン
AwItemAttributes::BY_LINEFONT	アイテム線種
AwItemAttributes::BY_LINEWEIGHT	アイテム線幅

ペン割り付けタイプには上記の他に AwItemAttributes::BY_COLOR があります。これはアイテム表示色をそのままペン番号とみなす方法で、このオブジェクトの割付表は参照しません。

AwPenAssignment オブジェクトは AwModel オブジェクトから得ます。

10.8.1 アクセッサ

現在のペン割り付けタイプを取得／設定します。デフォルト値は BY_CLASS です。
```
int  GetAssignmentType() const;
void SetAssignmentType(int type);
```
タイプ (AwItemAttributes::BY_COLOR - AwItemAttributes::BY_LINEWEIGHT)。

使用可能なペン数を取得／設定します。デフォルト値は 8 です。
```
int  GetMaxPen() const;
```

```
void SetMaxPen(int maxpen);
```
ペン数 (1 - AwPenAssignment::MAX_PEN_NUMBER)。

10.8.2 ペン割り付け表

以下のペン番号取得メソッドは、ペン割り付けタイプ が AwItemAttributes::BY_COLOR のときは、ペン番号を決定できないので、常に１を返します。

指定した AwItemAttributes に割り付けたペン番号を得ます。
```
int  GetPen(const AwItemAttributes& attr) const;
int  GetPen(int type, const AwItemAttributes& attr) const;
```
type　　ペン割り付けタイプ (AwItemAttributes::BY_ITEMTYPE - BY_LINEWEIGHT)。
attr　　AwItemAttributes オブジェクト。
戻り値　ペン番号を返します。

指定した AwSrAttributes に割り付けたペン番号を得ます。
```
int  GetPen(const AwSrAttributes& srAttr) const;
int  GetPen(int type, const AwSrAttributes& srAttr) const;
```
type　　ペン割り付けタイプ (AwItemAttributes::BY_ITEMTYPE - BY_LINEWEIGHT)。
srAttr　AwSrAttributes オブジェクト。
戻り値　ペン番号を返します。

指定した要素に割り付けたペン番号を得ます。
```
int  GetPen(int type, int no) const;
```
type　　ペン割り付けタイプ (AwItemAttributes::BY_ITEMTYPE - BY_LINEWEIGHT)。
no　　要素の番号 (1-)。たとえば type が BY_CLASS ならクラス番号です。
戻り値　ペン番号を返します。

以下のメソッドはペン割り付けの設定を行ないますが、現在のペン割り付けタイプは変更しません。
指定したペン割り付けタイプのひとつの要素のペン番号を設定します。
```
void SetPen(int type, int no, int pen);
```
type　　ペン割り付けタイプ (AwItemAttributes::BY_ITEMTYPE - BY_LINEWEIGHT)。
no　　要素の番号 (1-)。たとえば type が BY_CLASS ならクラス番号です。
pen　　ペン番号 (1 - GetMaxPen())。

指定したペン割り付けタイプの全ての要素に同じペン番号を設定します。
```
void SetPens(int type, int pen);
```
type　　ペン割り付けタイプ (AwItemAttributes::BY_ITEMTYPE - BY_LINEWEIGHT)。
pen　　ペン番号 (1 - GetMaxPen()))。

例）クラス５のペンを１番ペンにします。それ以外は変更しません。
```
AwPenAssignment* pAssign = pModel->GetPenAssignment();
if (pAssign)
  pAssign->SetPen(AwItemAttributes::BY_CLASS, 5, 1);
```

10.9 プリントアウト

● 関数一覧

関数名	機能
DrwO11	プロットファイルを作成する。（オフラインプロット）
DrwO12	プロット出力を行う。（オンラインプロット）

10.9.1 プロットファイルの作成

プロットファイルを作成する。(オフラインプロット)

【呼出し形式】

```
int Drw011(const char* name, int idrw)
```

【入力引数】

name	プロットファイル名 (Null-char terminated)。 プロットファイル名にファイル拡張子 ".PLT" を含める必要はない。
idrw	ドローイングレイアウト番号 0 :全ドローイングレイアウト

【戻り値】

0	: 正常終了
!0	: 異常終了

10.9.2 プロット出力

プロット出力を行う。(オンラインプロット)

【呼出し形式】

```
int Drw012(int idrw, const char* param, const char* log)
```

【入力引数】

idrw	ドローイングレイアウト番号 0 :全ドローイングレイアウト
param	バッチファイル oplot.bat に渡すパラメータ (Null-char terminated)。 複数のパラメータを指定するときはパラメータ間にスペースをいれる。 パラメータがないときは NULL ポインタ ((const char *)NULL) または空文字列 ("") を指定する。
log	"ERRLOG" : エラーログをファイル oplot.log に記録する。 "ERRCHK" : プロット出力の実行状態をウインドウに表示する。 その他 : エラーログもウインドウも必要ない場合は NULL ポインタまたは空文字列を指定する。

【戻り値】

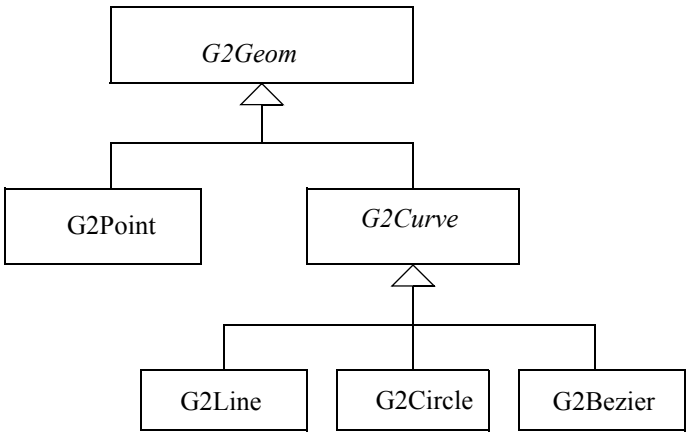
0	: 正常終了
!0	: 異常終了

第 11 章　幾何クラス

この章では下記の 2 次元の幾何クラスを説明します。

G2Math	ユーティリティ
G2Rect	矩形
G2Vector	ベクトル
G2Geom	幾何図形（抽象クラス）
G2Point	点
G2Curve	曲線（抽象クラス）
G2Line	線分 (G2Curve を継承)
G2Circle	円・円弧
G2Bezier	3 次 Bezier 曲線
G2Matrix	座標変換行列
G2PointArray	点配列
G2PtrArray	幾何図形オブジェクトのポインタ配列

線分、円・円弧、3 次 Bezier 曲線をまとめて「曲線」と呼びます。点と曲線をまとめて「幾何図形」と呼びます。幾何クラスの階層はこれを反映し、G2Line、G2Circle、G2Bezier クラスは G2Curve を継承し、G2Point、G2Curve クラスは G2Geom を継承しています。



それ以外のクラスは独立しています。G2PointArray は可変長の点列を表現するクラスで多角形としても使います。
G2PtrArray は幾何図形オブジェクトを格納するコンテナです。メソッドが計算した幾何図形オブジェクトを返すのに使用します。G2PtrArray はクラステンプレートです。実際に使用するのは G2PtrArray クラステンプレートをパラメタライズした G2CurveArray、G2LinePtrArray、G2CirclePtrArray テンプレートクラスです。

以下では特に断らない限り角度の単位はラジアンです。反時計回りの角度が正の値、時計回りの角度は負の値を持ちます。

曲線の左、右などの言い方が出てきますが曲線の始点から終点へ向かって曲線に沿って進むときの左右です。ベクトルの左、右も同様でベクトルの向きに対して言います。

11.1 G2Math クラス

このクラスは数学定数とクラスメソッドのみを持つユーティリティクラスです。

11.1.1 定数

定数名	説明
G2Math::CCw	反時計廻り
G2Math::Cw	時計廻り
G2Math::Narrow	劣角（せまい角度を取る）
G2Math::Left	曲線の左側
G2Math::Right	曲線の右側
G2Math::OnCurve	曲線上
G2Math::Inside	曲線の内側
G2Math::Outside	曲線の外側
G2Math::Start	曲線の始点
G2Math::End	曲線の終点
G2Math::Both	曲線の両端点（始点と終点）
G2Math::HalfPi	π/2
G2Math::Pi	π
G2Math::TwoPi	2π
G2Math::ToRadian	度をラジアンに単位変換
G2Math::ToDegree	ラジアンを度に単位変換
G2Math::DEps	最小値
G2Math::Eps	最小値（単精度実数用）

11.1.2 クラスメソッド

値を下限値と上限値の範囲に制限します。下限値より小さい値は下限値に、上限値より大きい値は上限値にクランプします。

```
static double Cramp(double lower, double value, double upper);
static int    Cramp(int    lower, int    value, int    upper);
```

lower　下限値。
value　値。
upper　上限値。

戻り値　lower <= value <= upper を満たす値を返します。

2 つの数値の小さい値を得ます。(STL の std::min() を使うこともできます)
```
static double Min(double d1, double d2);
static int    Min(int i1, int i2);
```
戻り値　小さい値を返します。

2 つの数値の大きい値を得ます。(STL の std::max() を使うこともできます)
```
static double Max(double d1, double d2);
static int    Max(int i1, int i2);
```
戻り値　大きい値を返します。

数値配列内の最小値を見つけます。(STL の std::min_elememt() を使うこともできます)
```
static int Min(const double values[], int count);
```
values　double の配列。
count　double の数。
戻り値　最小値の位置 (0 <= index < count) を返します。

最も近い整数に丸めます。
```
static double nint(double x);
```
戻り値　丸めた値を返します。
このメソッドはゼロに向かって丸めます。
```
G2Math::nint(-0.6) = -1.0
G2Math::nint(-0.5) = -1.0
G2Math::nint(-0.4) = -0.0
G2Math::nint( 0.4) =  0.0
G2Math::nint( 0.5) =  1.0
G2Math::nint( 0.6) =  1.0
```

60 進角度（度、分、秒）を十進角度に変換します。
```
static double Degree(double deg, double min, double sec);
```
deg　度（整数値）。
min　分（整数値）。
sec　秒。
戻り値　十進角度（単位は度）を返します。

十進角度を 60 進角度（度、分、秒）に変換します。
```
static void CalcDms(double degree, int resolution, int dms[]);
```
degrees　十進角度（単位は度）。
resolution　丸めの単位。1= 度、2= 分、3= 秒。
dms　結果を格納する配列。度 (dms[0]、分 (dms[1])、秒 (dms[2])。

角度を 0 - 360 度に正規化します。
```
static double NormalizeAngle(double degree);
```
degree　十進角度（単位は度）。
戻り値　0 - 360 度の角度を返します。

角度を計算します。
```
static double CalcAngle(double dx, double dy, int dir = Narrow);
```
dx　X 成分。
dy　Y 成分。
dir　角度測定方法。
- G2Math::CCw　反時計回りに測る。常に正の角度を得る (0 <= angle < 2π)。
- G2Math::Cw　時計回りに測る。常に負の角度を得る (-2π < angle <= 0)。
- G2Math::Narrow　劣角を得る (-π < angle <= π)。atan2(dy, dx) と同じ。

戻り値　角度（単位はラジアン）を返します。dx, dy が共に 0 のときは角度 0 を返します。また、角度の絶対値が G2Math::DEps 以下の時は角度 0 を返します。

長さを計算します。
```
static double CalcLength(double dx, double dy);
```
dx　X 成分。
dy　Y 成分。
戻り値　長さを返します。

11.2 G2Rect クラス

このクラスは矩形を表現します。矩形は移動、拡大はできますが、回転はできません。常に座標軸に平行で傾けることができないということです。
矩形は大きさが不定の状態（Invalid）と大きさを持つ状態 (Valid) があります。矩形が大きさを持つ状態ではその横幅、縦幅を得ることができます。1 点だけを追加した矩形は横幅、縦幅共に 0 です。水平な線分の両端点だけを追加したときは縦幅が 0 になります。垂直な線分の両端点だけを追加したときは横幅が 0 になります。このような特別な場合も Valid であることに注意してください。

11.2.1 コンストラクタ

```
G2Rect();
```
Invalid な矩形オブジェクトです。
```
G2Rect(const G2Point& p1, const G2Point& p2);
```
矩形の対角点 p1, p2 を与えます。
```
G2Rect(double x1, double y1, double x2, double y2);
```
矩形の対角点 (x1,y1)(x2,y2) を与えます。
```
G2Rect(const G2Point& center, double halfW, double halfH);
```
矩形の中心 center と大きさを与えます。横幅の半分 (halfW != 0)、縦幅の半分 (halfH != 0)。

11.2.2 状態

この矩形が有効か判定します。有効な矩形の横幅や縦幅は 0 であることもあります。
```
bool IsValid() const;
```
戻り値　有効な大きさを持つなら true を返します。

この矩形を無効にします。IsValid() メソッドが false を返す状態になります。
```
void Invalidate();
```

ほとんどのメソッドは有効な矩形でのみ機能します。無効な矩形でのメソッドの結果は不定です。

11.2.3 Getter

この矩形の最小座標（左下点）を得ます。
```
const G2Point& GetMin() const;
```
戻り値　左下点の参照を返します。
この矩形の最大座標（右上点）を得ます。
```
const G2Point& GetMax() const;
```
戻り値　右上点の参照を返します。

この矩形の横幅を得ます。
```
double GetWidth() const;
```
戻り値　この矩形の横幅を返します。幅は 0 であることもあります。
この矩形の縦幅を得ます。

```
double GetHeight() const;
```
戻り値　この矩形の縦幅を返します。幅は 0 であることもあります。

この矩形の中心点を得ます。
```
G2Point CalcCenter() const;
```
戻り値　矩形の中心点を返します。

この矩形の 4 隅の点を得ます。
```
void GetBoundPoints(G2Point points[], double clearance = 0.0) const;
```
points　4 隅の点を格納する配列。左下、右下、右上、左上の順。
clearance　clearance 分大きくしたり、小さくした矩形の 4 隅の点を計算します。負の値は矩形を小さくしますが、矩形の幅を 0 以下にするようなことはできません。-0.5 * Min(GetWidth(), GetHeight()) < clearance。

ふたつの矩形の共通部分を計算します。
```
G2Rect Intersection(const G2Rect& rect) const;
```
rect　もうひとつの矩形。
戻り値　共通部分を表す矩形を返します。共通部分がないとこの矩形は Invalid です。

この矩形の座標変換を計算します。この矩形の 4 隅の点を座標変換し、それらを含む矩形を計算します。矩形が回転したりするのではありません。
```
G2Rect CalcRect(const G2Matrix& matrix) const;
```
matrix　座標変換行列。
戻り値　座標変換した矩形を返します。

11.2.4 Setter

このオブジェクトに対角２点で矩形を設定します。結果は対角点の順序には依存しません。
```
void SetRect(const G2Point& p1, const G2Point& p2);
```
p1　対角点。
p2　もうひとつの対角点。
```
void SetRect(double x1, double y1, double x2, double y2);
```
x1, y1　対角点の座標。
x2, y2　もうひとつの対角点の座標。

このオブジェクトに指定した中心と大きさを持つ矩形を設定します。
```
void SetRect(const G2Point& center, double halfW, double halfH);
```
center　矩形の中心点。
halfW　横幅の半分 (halfW != 0)。
halfH　縦幅の半分 (halfH != 0)。

11.2.5 包含判定

この矩形が指定した点を含むか判定します。
```
bool Contains(const G2Point& point) const;
```
戻り値　この矩形が点を含むとき true を返します。

この矩形が点の X 座標を含むか判定します。Y 座標は無視します。
```
bool ContainsX(const G2Point& point) const;
```
戻り値　この矩形が点を含む（xmin <= point.x <= xmax）とき true を返します。

この矩形が点の Y 座標を含むか判定します。X 座標は無視します。
```
bool ContainsY(const G2Point& point) const;
```
戻り値　この矩形が点を含む（ymin <= point.y <= ymax）とき true を返します。

この矩形が引数で与えた矩形を含むか判定します。
```
bool Contains(const G2Rect& rect) const;
```
戻り値　この矩形が矩形 rect を含むとき true を返します。

この矩形が引数で与えた矩形と重なりを持つか判定します。
```
bool Intersects(const G2Rect& rect) const;
```
この矩形が矩形 rect と重なりを持つなら true を返します。

11.2.6 点の追加

この矩形に点や矩形を追加します。加えるものがこの矩形の内側にあるなら変化はありません。そうでなければ矩形はそれらを含むよう大きくなります。

この矩形に点を加えます。
```
void Add(const G2Point& point);
```
point　追加する点。
```
void Add(double x, double y);
```
x, y　追加する点の座標。

この矩形に点列を加えます。
```
void Add(int count, const G2Point points[]);
```
count　追加する点数。
points　追加する点の配列。

この矩形に矩形を加えます。
```
void Add(const G2Rect& rect);
```
rect　追加する矩形。Invalid な矩形は無視します。

11.2.7 拡大、移動

増分 (dx,dy) を与えてこの矩形の大きさを変更します。矩形は横幅 +2dx、縦幅 +2dy の大きさになります。増分を負の値にすると矩形は小さくなりますが、矩形の幅を 0 以下にすることはできません。
```
void InflateRect(double dx, double dy);
```
dx　横幅増分。-0.5 * GetWidth() < dx。
dy　縦幅増分。-0.5 * GetHeight() < dy。

この矩形の中心点を基準に矩形を拡大／縮小します。
```
void Magnify(double scale);
```
scale　倍率。G2Math::DEps <= scale。
```
void Magnify(double xscale, double yscale);
```
xscale　横幅倍率。G2Math::DEps <= xscale。
yscale　縦幅倍率。G2Math::DEps <= yscale。
この矩形を拡大／縮小します。矩形の中心点を基準にします。0 < xscale, yscale。

座標原点 (0,0) を基準にこの矩形を拡大／縮小します。
```
void Scale(double scale);
```
scale　倍率。G2Math::DEps <= scale。

この矩形を平行移動します。
```
void Translate(const G2Point& vector);
```
vector　移動量を与えます。矩形の座標 += 移動量となります。

11.2.8 クリッピング

クリッピングは矩形内部に含まれる部分だけを取り出すことです。

線分をクリップします。
```
int Clip(const G2Line& sline, G2Line* dline = NULL) const;
```
sline　クリップする線分。
dline　クリップされた線分。結果が不要なときはこの引数を省略するか null ポインタにします。
戻り値　結果を示す整数。
-1　線分は矩形の外側にある。dline には何も設定できなかった。
0　線分は矩形の内側にある。*dline = sline を設定した。
1 以上　線分の一部が矩形の内側にありクリッピングした。

円弧をクリップします。
```
int Clip(const G2Circle& circle, G2Circle arcs[], double dmin) const;
```
circle　クリップする円弧。
arcs　クリップされた円弧を格納する配列。
dmin　クリップされた円弧の最小長さ (0 < dmin)。これより短い円弧は取り除きます。
戻り値　出力配列に格納した円弧の数 (0-5)。

Bezier 曲線をクリップします。
```
int Clip(const G2Bezier& bezier, G2Bezier arcs[], double dmin) const;
```
bezier　クリップする Bezier 曲線。
arcs　クリップされた Bezier 曲線を格納する配列。
dmin　クリップされた Bezier 曲線の最小長さ (0 < dmin)。これより短い曲線は取り除きます。
戻り値　出力配列に格納した曲線の数 (0-3)。

反時計回りの単純多角形をクリップします。
```
int Clip(int npnt, G2Point points[], int maxpnt) const;
```
npnt　単純多角形の点数（3 -)。
points　単純多角形を格納する配列 points[maxpnt]。クリップした多角形はここに格納します。
maxpnt　配列 points のサイズ (npnt <= maxpnt)。
戻り値　クリップ後の多角形の点数（0 - maxpnt)。
クリップ後の多角形の点数が増えることもあります。例えばひし形（4 角形）をクリップした結果が 8 角形になることもあります。クリップ後の多角形の点数が maxpnt を超えるときは計算を中断し、0 を返します。エラーの場合でも入力配列 points の内容は変更されています。

例）三つの線分を矩形でクリップする。
```
   const G2Rect rect(-10.0, -10.0, 10.0, 10.0);
   const G2Line lines[] = {
           G2Line(G2Point(-5.0, 15,0), G2Point( 0.0, 20.0)),     // 矩形の外側
           G2Line(G2Point(-5.0,  0.0), G2Point( 0.0,  5.0)),     // 矩形の内側
           G2Line(G2Point(-5.0,-15.0), G2Point(15.0,  5.0))      // 矩形と交差
         };
   G2Line out;
   for (int j = 0; j < sizeof(lines) / sizeof(lines[0]); ++j) {
     if (rect.Clip(lines[j], &out) >= 1) {
       printf("clipped line %g %g %g %g¥n",
              out.GetStartPoint().x, out.GetStartPoint().y,
              out.GetEndPoint().x, out.GetEndPoint().y);
   }
```
lines[2] だけが クリップされ、結果は (0,-10),(10,0) となります。

11.3 G2Vector クラス

この クラス は 2D ベクトルを表現するクラスです。ベクトルは向きと長さを持ちます。向きは単位ベクトル、長さは 0 以上の値です。長さが 極小のベクトルは無効とみなし、IsValid() メソッドは false を返します。

11.3.1 定数オブジェクト

```
static const G2Vector Xunit;
```
正の X 軸の単位ベクトル (1, 0) を持つオブジェクト。
```
static const G2Vector Xunit;
```
正の Y 軸の単位ベクトル (0, 1) を持つオブジェクト。

11.3.2 コンストラクタ

```
G2Vector();
```
単位ベクトル (1, 0)、長さは 1。
```
G2Vector(double angle);
```
角度 (angle) のベクトル。長さは 1。
```
G2Vector(double dx, double dy);
```
X 成分 dx、Y 成分 dy のベクトル。
```
G2Vector(const G2Point& ps, const G2Point& pe);
```
ps から pe へ向うベクトル（pe - ps)。

引数から計算したベクトル長さが G2Math::DEps 未満の時は、単位ベクトル (1, 0)、長さ 0 にします。そのようなベクトルは IsValid() が false を返します。

11.3.3 Getter

ベクトルが有効か判定します。
```
bool IsValid() const;
```
戻り値　このベクトルの長さが G2Math::DEps 以上であれば true を返します。

単位ベクトルの成分を得ます。
```
double Ux() const;
```
戻り値　このベクトルの X 成分を返します。(0 <= |Ux()| <= 1)。
```
double Uy() const;
```
戻り値　このベクトルの Y 成分を返します。(0 <= |Uy()| <= 1)。

ベクトルの長さを得ます。
```
double GetLength() const;
```
戻り値　このベクトルの長さを返します。(0 <= GetLength())

11.3.4 Setter

このオブジェクトに指定した角度の単位ベクトルを設定します（ベクトル長さは 1 になります)。
```
void SetVector(double angle);
```
angle　　ベクトルの角度。

このオブジェクトに指定した X 成分、Y 成分のベクトルを設定します。ベクトル長さは X 成分、Y 成分から計算します。
```
void SetVector(double dx, double dy);
```
dx　　X 成分。

dy　　Y 成分。

このオブジェクトに指定した始点から終点へ向うベクトルを設定します。ベクトル長さは 2 点間の距離になります。
```
void SetVector(const G2Point& ps, const G2Point& pe);
```
ps　　始点。
pe　　終点。

このオブジェクトのベクトルの長さを変更します。
```
void SetLength(double length);
```
length　　ベクトルの長さ。length が負のときは、このベクトルの向きを反転し長さは length の絶対値にします。このベクトルの長さが 0 より小さくなることはありません。

11.3.5 演算子

ベクトルの和、差を計算する演算子 (+、-) があります。これらの演算子は計算結果を新しい G2Vector オブジェクトで返します。これに対応する代入を伴う演算子 (+=、-=) は演算子を呼び出したオブジェクトを計算結果で書き換えます。

ベクトルの和 (this + vec) を計算します。
```
G2Vector operator+(const G2Vector& vec) const;
```
vec　　加えるベクトル。
戻り値　　ベクトルの和のオブジェクトを返します。

ベクトルの加算。このオブジェクトにベクトル vec を加えます (this = this + vec)。
```
G2Vector& operator+=(const G2Vector& vec);
```
vec　　加えるベクトル。
戻り値　　このオブジェクトの参照を返します。

ベクトルの差 (this - vec) を計算します。
```
G2Vector operator-(const G2Vector& vec) const;
```
vec　　引くベクトル。
戻り値　　ベクトルの差のオブジェクトを返します。

ベクトルの減算。このオブジェクトからベクトル vec を引きます (this = this - vec)。
```
G2Vector& operator-=(const G2Vector& vec);
```
vec　　引くベクトル。
戻り値　　このオブジェクトの参照を返します。

11.3.6 回転

このベクトルを左に 90 度回転したベクトルを計算します。
```
G2Vector TurnLeft() const;
```

このベクトルを右に 90 度回転したベクトルを計算します。
```
G2Vector TurnRight() const;
```

このベクトルの向きを逆にした（180 度回転した）ベクトルを計算します。
```
G2Vector Reverse() const;
```

このベクトルを指定した角度だけ回転したベクトルを計算します。
```
G2Vector Rotate(double angle) const;
```
angle　　回転角度。

11.3.7 ベクトル上の点

ベクトル上の点を計算します。
G2Point PointAt(double lprm) const;
　戻り値　　ベクトル上の点を計算します。point = (lprm * Ux(), lprm * Uy())。

11.3.8 向きの比較

2 つのベクトルの向きの関係を調べるメソッドです。ベクトルの長さは無関係です。引数 eps は許容差です。

ベクトルが同じ向きか判定します。
bool IsSameDirection(const G2Vector& v, double eps) const;
　戻り値　　同じ向きなら true を返します。

ベクトルが逆向きか判定します。
bool IsReversedDirection(const G2Vector& v, double eps) const;
　戻り値　　逆向きなら true を返します。

ベクトルが平行か判定します。
bool IsParallel(const G2Vector& v, double eps) const;
　戻り値　　平行なら（向きは逆かもしれない）true を返します。

ベクトルが直交するか判定します。
bool IsPerpendicular(const G2Vector& v, double eps) const;
　戻り値　　直交するなら true を返します。

ベクトルの向きを判定します。
nt Side(const G2Vector& v, double eps) const;
　戻り値　　このベクトルを基準にベクトル v がどちら側を向いているかを返します。
　　G2Math::Left　　v は左側。
　　G2Math::OnCurve　平行。
　　G2Math::Right　　v は右側。

11.3.9 内積

2 次元ベクトルの内積は次式で計算します。
　dot = (dx1, dy1)・(dx2, dy2) = dx1 * dx2 + dy1 * dy2.
ベクトル v を単位ベクトル e に正射影した正射影ベクトルを e’ とします。内積 (e・v) の絶対値は正射影ベクトル e’ の長さに相当します。

この単位ベクトルと引数のベクトルの内積を計算します。
double CalcXt(const G2Point& p) const;
　p　　ベクトル (p.x, p.y)。
double CalcXt(double dx, double dy) const;
　dx, dy　　ベクトル (dx,dy)。

この単位ベクトルを 90 度左に回転したベクトルと引数のベクトルの内積を計算します。
double CalcYt(const G2Point& p) const;
　p　　ベクトル (p.x, p.y)。
double CalcYt(double dx, double dy) const;
　dx, dy　　ベクトル (dx,dy)。

11.3.10 角度計算

このベクトルの角度を計算します。
double GetAngle(int dir = G2Math::Narrow) const;
　dir　　角度測定方法。
　　G2Math::CCw　　反時計回りに測る。常に正の角度を得る (0 <= angle < 2π)。
　　G2Math::Cw　　時計回りに測る。常に負の角度を得る (-2π < angle <= 0)。
　　G2Math::Narrow　　劣角を得る (-π < angle <= π)。
　戻り値　このベクトルの (+X 軸からの) 角度を返します。

ふたつのベクトル成す角度を計算します。
double CalcAngle(const G2Vector& v, int dir = G2Math::Narrow) const;
　v　　ベクトル。
　dir　　角度測定方法。
　　G2Math::CCw　　反時計回りに測る。常に正の角度を得る (0 <= angle < 2π)。
　　G2Math::Cw　　時計回りに測る。常に負の角度を得る (-2π < angle <= 0)。
　　G2Math::Narrow　　劣角を得る (-π < angle <= π)。
　戻り値　このベクトルからベクトル v への角度を返します。

11.4 G2Geom クラス

このクラスは２次元幾何図形を表現する最上位の抽象基底クラスです。G2Geom クラスは全ての幾何図形に共通するメソッドを持ちます。

11.4.1 定数

定数名	説明
G2Geom::POINT	点
G2Geom::LINE	線分
G2Geom::CIRCLE	円
G2Geom::BEZIER	Bezier 曲線
G2Geom::CONIC	円錐曲線（予約）
G2Geom::VECTOR	ベクトル

11.4.2 図形のタイプ

幾何図形のタイプを得ます。
int GetId() const;
　戻り値　このオブジェクトの図形タイプを返します。

図形タイプから具象クラスがわかります。図形タイプごとに具象クラスがあります。次のコードフラグメントは図形の抽象クラスを具象クラスにダウンキャストする例です。

```
// ここでは pGeom は const G2Geom* とします。
if (pGeom->GetId() == G2Geom::POINT) {
  const G2Point* pPoint = static_cast<const G2Point*>(pGeom);
  // G2Point クラスのメソッドを使うことができる。
} else if (pGeom->GetId() == G2Geom::LINE) {
```

```
    const G2Line* pLine = static_cast<const G2Line*>(pGeom);
    // G2Line クラスのメソッドを使うことができる。
}
```

図形タイプが G2Geom::POINT であれば、これは 2D の点で、それの具象クラスは G2Point クラスです。
次の文は static_cast を使って const G2Geom* を const G2Point* にダウンキャストします。

```
const G2Point* pPoint = static_cast<const G2Point*>(pGeom);
```

このようにして得たポインタ pPoint を使うと G2Point クラスのメソッドを使うことができます。
このとき、間違った具象クラスにダウンキャストしないようにプログラミングしなければなりません。
dynamic_cast を使うと間違った具象クラスにダウンキャストしようとすると null ポインタを返します。
dynamic_cast は実行時にクラスの継承を調べるので正確ですが、実行時の負荷が大きくなります。

GetId() メソッドを使えばできることですが、簡便のために用意されたメソッドがあります。
次のメソッドは図形オブジェクトが曲線タイプ（G2Geom::LINE、G2Geom::CIRCLE または G2Geom::BEZIER）か判定します。

```
bool IsCurve() const;
```

戻り値　このオブジェクトが曲線タイプなら true を返します。

11.4.3 オブジェクトのコピイ

G2Geom クラスは抽象クラスなので、オブジェクトのコピイを作るには、具象クラスを調べてコピイしなければなりません。これを簡単に行えるように、図形オブジェクトをコピイするメソッドがあります。

```
G2Geom* Clone() const;
```

戻り値　具象クラスのオブジェクトのコピイを作りそのポインタを返します。このメソッドで得たオブジェクトは不要になった時点で delete しなければなりません。

ここで注意をひとつ。G2Geom クラスの代入演算子 (=) は使用できません。G2Geom* のポインタ p Geom1、pGeom2 に対して次のような代入を行うとコンパイルエラーになるようにしています。

```
*pGeom2 = *pGeom1; // オブジェクトの内容をコピイしたいができない。
```

この代入文は、G2Geom クラスのデータだけをコピイし具象クラスで定義したデータはコピイしないのに、あたかも具象クラスのデータもコピイすると勘違いしやすいからです。
ポインタ変数の代入はできます。次の文は pGeom2 が p Geom1 と同じオブジェクトを指すようにします。

```
pGeom2 = pGeom1; // ポインタの代入。
```

もし p Geom1 と pGeom2 の指すオブジェクトの具象クラスが同じなら、次のようにダウンキャストすれば代入演算子が使えます（オブジェクトの内容をコピイ）。

```
static_cast<G2Line&>(*pGeom2) = static_cast<G2Line&>(*pGeom1); // 両方 G2Line オブジェクト
```

11.4.4 判定

オブジェクトの有効性判定、ふたつのオブジェクトの幾何的等価性を判定するメソッドがあります。これらの判定基準は図形タイプに依存します。

このオブジェクトが幾何的に有効なデータを持つか判定します。

```
bool IsValid() const;
```

戻り値　このオブジェクトが有効なデータを持つとき true を返します。

ふたつのオブジェクトが幾何的に等価か判定します。

```
bool Equals(const G2Geom& geom, double tol) const;
```

geom　比較する G2Geom オブジェクト。
tol　座標や長さを比較するときの許容値。
戻り値　与えられた許容値以内で等価であるとき true を返します。

11.4.5 座標変換

この幾何図形を平行移動します。
```
void Translate(const G2Point& vector);
```
vector　移動量を与えます。幾何図形の座標 += 移動量となります。

この幾何図形を座標変換します。
```
void Transform(const G2Matrix& matrix);
```
matrix　座標変換行列。

この幾何図形を逆座標変換します。
```
void InverseTransform(const G2Matrix& matrix);
```
matrix　座標変換行列。

11.4.6 最短距離

ふたつの幾何図形の最短距離を計算するメソッドがあります。図形が線分または円弧の場合は、これらを有限長で計算するか無限長とみなして計算するかを選択できます。円弧の場合、無限長とは円のことです。

この幾何図形と指定した点との最短距離を計算します。
```
double Distance(const G2Point& point, int iswt = 0, G2Point* pon = NULL, double* lprm = NULL) const;
```
iswt　選択肢。
- 0　無限長。線分なら直線、円弧なら円として処理します。
- 1　有限長。

point　最短距離を計算する相手の点。
pon　この幾何図形上の最短点を格納する G2Point オブジェクトへのポインタ。不要なときは省略するか null ポインタを渡します。
lprm　の幾何図形上の最短点における曲線パラメータ。不要なときは省略するか null ポインタを渡します。
戻り値　最短距離 (0 <= distance)。

この幾何図形と引数の幾何図形との最短距離を計算します。
```
int Measure(int iswt, const G2Geom& geom, double* dist, G2Point pons[]) const;
```
iswt　選択肢。
- 0　無限長。線分なら直線、円弧なら円として処理します。
- 1　有限長。

geom　最短距離を計算する相手の幾何図形。
dist　最短距離 (0 <= *dist)。
pons　最短点を格納する G2Point オブジェクトの配列。pons[0] はこの幾何図形上の点、pons[1] は相手の幾何図形上の点を格納します。
戻り値　最短距離が求まったとき 0 を返します (0 -)。

11.4.7 共通接線

線分オブジェクトには共通接線はありません。点オブジェクトは共通接線の端点（始点または終点）になります。

この幾何図形と引数の幾何図形の共通接線を計算します。接線の始点はこの幾何図形上にあり、終点は相手の幾何図形上にあります。
```
int Tangent(const G2Geom& geom, G2LinePtrArray* pLines) const;
```
geom　接線を計算する相手の幾何図形。
pLines　接線を追加する G2LinePtrArray オブジェクトへのポインタ。
戻り値　追加した接線の数を返します (0 -)。

指定した向きの接線を計算します。接線の始点はこの幾何図形上にあります。
```
int Tangent(const G2Vector& vector, G2LinePtrArray* pLines) const;
```
vector　接線の向き。
pLines　接線を追加する G2LinePtrArray オブジェクトへのポインタ。
戻り値　追加した接線の数を返します (0 -)。

11.4.8 共通接円弧

点オブジェクトは共通接円弧の端点（始点または終点）になります。

この幾何図形と引数の幾何図形の共通接円弧を計算します。円弧の始点はこの幾何図形上にあり、終点は相手の幾何図形上にあります
```
int Fillet(const G2Geom& geom, double radius, G2CirclePtrArray* pArcs) const;
```
geom　接円を計算する相手の幾何図形。
rasius　接円の半径 (0 < radius)。
pArcs　接円弧を追加する G2CirclePtrArray オブジェクトへのポインタ。
戻り値　追加した接円弧の数を返します (0 -)。

指定した始点を持つ接円弧を計算します。接円弧の終点はこの幾何図形上にあります。
```
int Fillet(const G2Point& start, const G2Vector& vector, G2CirclePtrArray* pArcs) const;
```
start　接円弧の始点。この幾何図形から離れた点を与えます。
vector　接円弧の始点における接線。
pArcs　接円弧を追加する G2CirclePtrArray オブジェクトへのポインタ。
戻り値　追加した接円弧の数を返します (0 -)。

この幾何図形の接円を計算します。接円の端点 (始点 == 終点) はこの幾何図形上にあります。
```
int TangentCircles(const G2Point& center, G2CirclePtrArray* pCirs, double tol) const;
```
center　接円の中心。この幾何図形から離れた点を与えます。
pCirs　接円を追加する G2CirclePtrArray オブジェクトへのポインタ。
tol　許容値。
戻り値　追加した接円の数を返します (0 -)。

11.5 G2Curve クラス

このクラスは２次元曲線を表現する抽象基底クラスです。G2Geom クラスを継承し、全ての曲線に共通するメソッドを持ちます。以下では基底クラスから継承したメソッドは除き、このクラスで追加するメソッドを主に説明します。

曲線上の点を特定するにはパラメータを使います。ここでは曲線の始点からの曲線長をパラメータに採用しています。曲線の始点から終点に向かう曲線に沿った長さです。例えば線分の始点から 10mm の位置の点といったように使います。線分ではパラメータの値が負であれば、線分の向きと逆の方に点を取ります。円ではパラメータの範囲は円周の長さ以内です (0 <= lprm <= 2πr、r は円の半径)。Bezier 曲線ではパラメータの範囲は 0 <= lprm <= 曲線長です。この定義域以外では点は不定です。

G2Curve クラスは抽象クラスなので、オブジェクトのコピイを作るには、具象クラスを調べてコピイしなければなりません。これを簡単に行えるように、曲線オブジェクトをコピイするメソッドがあります。
```
G2Curve* Clone() const;
```
このオブジェクトをコピイをします。このメソッドで得た G2Curve オブジェクトは不要になった時点で delete しなければなりません。

11.5.1 曲線諸元

この曲線の始点を得ます。
```
const G2Point& GetStartPoint() const;
```
この曲線の終点を得ます。
```
const G2Point& GetEndPoint() const;
```

この曲線の長さを得ます。
```
double GetLength() const;
```

この曲線を包含する最小矩形を計算します。
```
G2Rect GetBounds() const;
```

この曲線上の点に対応するパラメータを計算します。
```
double LparamAt(const G2Point& point) const;
```
point　曲線上の点。
戻り値　曲線のパラメータ。

この曲線の指定したパラメータにおける点を計算します。
```
G2Point PointAt(double lprm) const;
```
lprm　曲線のパラメータ。
戻り値　パラメータ lprm における点。

この曲線の指定したパラメータにおける微分を計算します。
```
G2Vector DerivativeAt(double lprm) const;
```
lprm　曲線のパラメータ。
戻り値　パラメータ lprm における微分（接ベクトル、速度ベクトル)。

この曲線の指定したパラメータにおける二階微分を計算します。
```
G2Vector Derivative2At(double lprm) const;
```
lprm　曲線のパラメータ。
戻り値　パラメータ lprm における二階微分（加速度ベクトル)。

この曲線の端点における接ベクトルを計算します。接ベクトルは指定した端点からもう一方の端点に向かって計算します。始点側では曲線の進行方向に、終点側では曲線の進行方向とは逆方向に向って測ります。
```
G2Vector TangentAtEnd(int which) const;
```
which　G2Math::Start は始点、G2Math::End は終点を指定します。
戻り値　接ベクトル。

この曲線上の点における法線を計算します。
```
bool CalcNormal(const G2Point& point, G2Line* normal) const;
```
point　曲線上の点。
normal　法線を格納する線分。始点は曲線上の点 point です。
戻り値　法線を計算できたとき true を返します。

この曲線の向きを反転します。
```
void Reverse();
```

11.5.2 判定

指定した点が曲線上にあるか判定します。
```
bool IsOnCurve(const G2Point& point, double tol, bool bExactly = false) const;
```
point　判定する点。
tol　許容差。
bExactly　判定方法。true なら点が厳密に曲線上にあるか判定します。false は点が、曲線上より緩いある範囲内にあるか判定します。ある範囲とは、線分では両端点を通る直交直線の間、

円弧ではその円弧の中心点と始点から終点の間の成す開いたパイをいいます。点がその範囲に入れば true を返します。

戻り値　点が曲線上にあれば true を返します。

パラメータが定める点が曲線上にあるか判定します。

```
bool ContainsLength(double lprm, double tol) const;
```

lprm　曲線のパラメータ。

tol　許容差。

戻り値　0 <= lprm <= 曲線長なら true を返します。

指定した点が曲線のどちら側にあるか判定します。

```
int Side(const G2Point& point) const;
```

point　判定する点。

戻り値　判定結果を返します。

- G2Math::Left　曲線の左。
- G2Math::OnCurve　曲線上または判定不能。
- G2Math::Right　曲線の右。

11.5.3 点の計算

指定した点をこの曲線に正射影した曲線上の点（投影点）を計算します。投影点の最大数は曲線に依存します。線分は 1 個、円は 2 個、3 次 Bezier 曲線では 5 個が最大数です。

```
int Projection(const G2Point& point, G2Point pons[], double lprms[], int iswt = 1) const;
```

point　曲線に投影する点。

pons　投影点を格納する配列。

lprms　投影点におけるこの曲線パラメータを格納する配列。

iswt　円、Bezier 曲線の場合の投影点選択肢。

- 0　全ての投影点を計算する。
- 1　point に最も近い投影点のみ計算する。
- 2　point から最も遠い投影点のみ計算する (円のみ)。

戻り値　出力配列に格納した投影点の数 (0-)。

ふたつの曲線の交点を計算します。交点の最大数は曲線の組み合わせに依存します。線分と線分の交点は 1 個、3 次 Bezier 曲線と 3 次 Bezier 曲線では 9 個が最大数です。

```
int Intersection(const G2Curve& curve, G2Point pins[], double lprms[][2]) const;
```

curve　もうひとつの曲線オブジェクト。

pins　交点を格納する配列。

lprms　交点位置の曲線パラメータを格納する配列。

- lprms[][0]　この曲線のパラメータ。
- lprms[][1]　引数の曲線のパラメータ。

戻り値　出力配列に格納した交点の数 (0-9)。

この曲線上の等間隔点列を計算します。このオブジェクトが Bezier 曲線なら近似の等間隔点です。

```
int PointSamplerEsp(int count, G2PointArray* points) const;
```

count　計算する点数 (1 -)。

- 1　曲線の中点を計算する。
- 2　曲線の始点、終点を計算する。
- 3 以上　等間隔点列を計算する (曲線の両端点を含む)。

points　点を追加する G2PointArray オブジェクトへのポインタ。このオブジェクトに追加できる点数が count 以上でなければなりません。

戻り値　追加した点の数 (0 - count)。

曲線上の点列を結ぶ折れ線と曲線との差が指定した許容差に収まるような点列を計算します。詳細は G2Line、G2Circle、G2Bezier クラスの PointSamplerTol() を参照してください。

```
int PointSamplerTol(int mode, double tol, int maxtry, G2PointArray* points) const;
```

mode　許容値の種類。
- 1　相対値。(0.0001 <= 弧高 / 弦長 <= 0.25)
- 2　絶対値。(弧高)

tol　許容値の種類に従った許容値。
maxtry　繰り返しの打ち切り回数。
points　点を追加する G2PointArray オブジェクトへのポインタ。
戻り値　追加した点の数 (0 -)。

11.5.4 分割／連結

この曲線を曲線上の点で二分割した曲線を計算します。

```
int SplitAt(const G2Point& point, double dmin, G2Curve** first, G2Curve** second) const;
```

point　曲線上の点。
dmin　許容値。
first　分割後の前半の曲線へのポインタ。
second　分割後の後半の曲線へのポインタ。
戻り値　結果を表す整数。
- 0　ふたつに分割できた。
- 1　分割位置が始点と一致。*first == NULL
- 2　分割位置が終点と一致。*second == NULL
- 3　分割できない。*first == NULL, *second == NULL

first、second の指すオブジェクトは不要となった時点で delete しなければなりません。

ふたつの曲線を連結します。ふたつの曲線のタイプが同じでなければ連結できません。この曲線をふたつの曲線を連結した曲線で置き換えます。

```
bool Merge(const G2Curve& second, double tol);
```

curve　もうひとつの曲線オブジェクト。
tol　許容値。
戻り値　この曲線がふたつの曲線を連結した曲線で置き換えられたとき true を返します。

11.6 G2Point クラス

このクラスは２次元の点を表現するクラスです。G2Geom クラスを継承します。以下では基底クラスから継承したメソッドは除き、このクラスで追加するメソッドを主に説明します。

11.6.1 定数オブジェクト

```
static const G2Point Origin;
```

座標 (0, 0) を持つオブジェクト。

11.6.2 コンストラクタ

```
G2Point();
```

座標 (0, 0) の点オブジェクト。

```
G2Point(double x, double y);
```

座標 (x, y) の点オブジェクト。

11.6.3 Getter

このオブジェクトは２次元座標 (x, y) を持ちます。メンバー変数 x, y は public で、直接アクセスできます。

11.6.4 Setter

このオブジェクトに座標 (x, y) を設定します。
```
void Set(double x, double y);
```

11.6.5 判定

このオブジェクトが有効な点かどうか判定します。座標が -1.0e10 <= x, y <= 1.0e10 であれば有効な点です。
```
bool IsValid() const;
```
戻り値　有効な点なら true を返します。

ふたつの点が等価か判定します。
```
bool Equals(const G2Point& point, double tol) const;
```
point　比較する G2Point オブジェクト。
tol　座標を比較するときの許容値。
戻り値　与えられた許容値以内で等価であるとき true を返します。

引数の二点のどちらがこの点に近いか判定します。
```
int SelectNear(const G2Point& p1, const G2Point& p2) const;
```
戻り値　0 : p1 がこの点に近い、1 : p2 がこの点に近い。

11.6.6 角度

この点を中心に点 ps から点 pe に向かう角度を計算します。
```
double CalcAngle(const G2Point& ps, const G2Point& pe,  int dir = G2Math::Narrow) const;
```
ps　角度始点。
pe　角度終点。
dir　角度測定方向。
G2Math::CCw　反時計回り（0 <= angle < 2π）
G2Math::Cw　時計回り（-2π < angle <= 0）
G2Math::Narrow　劣角。（-π < angle <= π）
戻り値　角度。

11.6.7 演算子

点の和、差、スカラー積、スカラー除を計算する演算子 (+、-、*、/) があります。これらの演算子は計算結果を新しい G2Point オブジェクトで返します。これに対応する代入を伴う演算子 (+=、-=、*=、/=) は演算子を呼び出したオブジェクトを計算結果で書き換えます。

点座標の和 (this + point) を計算します。
```
G2Point operator+(const G2Point& point) const;
```
point　加える点。
戻り値　点座標の和のオブジェクトを返します。

点座標の加算。このオブジェクトに点 point を加えます (this = this + point)。
```
G2Point& operator+=(const G2Point& point);
```

point　　加える点。
戻り値　　このオブジェクトの参照を返します。

点座標の差 (this - point) を計算します。
```
G2Point operator-(const G2Point& point) const;
```
point　　引く点。
戻り値　　点座標の差のオブジェクトを返します。

点座標の減算。このオブジェクトから点 point を引きます (this = this - point)。
```
G2Point& operator-=(const G2Point& point);
```
point　　引く点。
戻り値　　このオブジェクトの参照を返します。

点座標のスカラー積 (this * s) を計算します。
```
G2Point operator*(double s) const;
```
s　　乗数。
戻り値　　点座標のスカラー積のオブジェクトを返します。

点座標のスカラー積。このオブジェクトの座標にスカラー s を掛けます (this = this * s)。
```
G2Point& operator*=(double s);
```
s　　乗数。
戻り値　　このオブジェクトの参照を返します。

点座標のスカラー除 (this / s) を計算します。
```
G2Point operator/(double s) const;
```
s　　除数。s != 0.0。
戻り値　　この点座標のスカラー除のオブジェクトを返します。

点座標のスカラー除。このオブジェクトの座標をスカラー s で割ります (this = this / s)。
```
G2Point& operator/=(double s);
```
s　　除数。s != 0.0。
戻り値　　このオブジェクトの参照を返します。

以下に簡単な例を示します。
```
G2Point p1( 5.0,  4.0);
G2Point p2( 2.0,  3.0);
G2Point p3;
p3 = p1 + p2;    // p3 (7.0, 7.0)
p3 = p1 - p2;    // p3 (3.0, 1.0)
p3 = p1 * 0.5;   // p3 (2.5, 2.0)
p3 = 0.5 * p1;   // コンパイルエラーになります。
p3 = p1 / 2.0;   // p3 (2.5, 2.0)
p1 += p2;        // p1 (7.0, 7.0)
p1 -= p2;        // p1 (5.0, 4.0)
p3 = (p1 + p2) / 2.0;    // p3 (3.5, 3.5) = p1 と p2 の中点。
```

11.6.8　点にベクトルを加算

この点にベクトル (s * v) を加えた点を計算します。
```
G2Point Add(double s, const G2Vector& v) const;
```
s　　ベクトルの長さ。
v　　ベクトル。
戻り値　　点 (this + s * v) オブジェクトを返します。

この点にベクトル v を加えた点を計算します。Add(v.GetLength(), v) の省略形です。

```
G2Point Add(const G2Vector& v) const;
```
v　　ベクトル。
戻り値　　点 (this + v.GetLength() * v) オブジェクトを返します。

ふたつの点の中点を計算します。
```
G2Point CalcMidPoint(const G2Point& p) const;
```
p　　点。
戻り値　　この点と引数の点 p の中点を返します。

11.6.9 二点間距離

２点の距離を計算します。
```
double CalcDistance(const G2Point& point) const;
```

２点の距離の自乗を計算します。
```
double CalcSquareDistance(const G2Point& point) const;
```

２点の L1 距離を計算します。L1 距離は 2 点の座標の差分 dx, dy の絶対値の和です。
L1 = | this.x - point.x | + | this.y - point.y |。
```
double CalcL1Distance(const G2Point& point) const;
```

２点の L ∞距離を計算します。
L ∞距離は 2 点の座標の差分 dx, dy, dz の絶対値の最も大きいものです。
L ∞ = Max(| this.x - point.x |, | this.y - point.y)。
```
double CalcLinfDistance(const G2Point& point) const;
```

11.6.10 最短距離

この点オブジェクトと幾何図形（G2Point、G2Line、G2Circle、G2Bezier）間の最短距離を計算するメソッドです。
```
int Measure(int iswt, const G2Point& point, double* dist, G2Point pons[]) const;
int Measure(int iswt, const G2Line& line, double* dist, G2Point pons[]) const;
int Measure(int iswt, const G2Circle& cir, double* dist, G2Point pons[]) const;
int Measure(int iswt, const G2Bezier& bzc, double* dist, G2Point pons[]) const;
```
iswt　　選択肢。
　0　　無限長。線分なら直線、円弧なら円として処理します。
　1　　有限長。
dist　　最短距離 (0 <= *dist)。
pons　　最短点を格納する G2Point オブジェクトの配列。pons[0] はこの点オブジェクト、pons[1] は第 2 引数の幾何図形上の点を格納します。
戻り値　最短距離が求まったとき 0 を返します (0 -)。

11.6.11 共通接線

この点オブジェクトと幾何図形（G2Point、G2Circle、G2Bezier）の共通接線を計算するメソッドです。接線の始点は点オブジェクトで、終点はもうひとつの幾何図形上にあります。
```
int Tangent(const G2Point& point, G2LinePtrArray* pLines) const;
int Tangent(const G2Circle& cir, G2LinePtrArray* pLines) const;
int Tangent(const G2Bezier& bzc, G2LinePtrArray* pLines) const;
```
pLines　線分を追加する G2LinePtrArray オブジェクトへのポインタ。
戻り値　追加した線分の数を返します (0 -)。

11.6.12 共通接円弧

この点オブジェクトと幾何図形（G2Point、G2Line、G2Circle、G2Bezier）の共通接円弧を計算するメソッドです。円弧の始点はこのオブジェクトで終点はもうひとつの幾何図形上にあります。

```
int Fillet(const G2Point& point, double radius, G2CirclePtrArray* pArcs) const;
int Fillet(const G2Line& line, double radius, G2CirclePtrArray* pArcs) const;
int Fillet(const G2Circle& cir, double radius, G2CirclePtrArray* pArcs) const;
int Fillet(const G2Bezier& bzc, double radius, G2CirclePtrArray* pArcs) const;
```

radius　接円の半径 (0 < radius)。
pArcs　円弧を追加する G2CirclePtrArray オブジェクトへのポインタ。
戻り値　追加した円弧の数を返します (0 -)。

11.7 G2Line クラス

このクラス は２次元の線分を表現するクラスです。G2Curve クラスを継承します。以下では基底クラスから継承したメソッドは除き、このクラスで追加するメソッドを主に説明します。

11.7.1 コンストラクタ

```
G2Line();
```
始点、終点ともに (0, 0) の線分。
```
G2Line(const G2Line& line);
```
コピイコンストラクタ。
```
G2Line(const G2Point& start, const G2Point& end);
```
始点 start から終点 end へ向かう線分オブジェクト。
```
G2Line(const G2Point& start, const G2Vector& vector);
```
始点 start、向きが vector の線分オブジェクト。

11.7.2 Getter

この線分の向きを得ます。
```
G2Vector GetVector() const;
```
戻り値　G2Vector オブジェクトを返します。

11.7.3 Setter

このオブジェクトに始点 start から終点 end へ向かう線分を設定します。
```
void SetLine(const G2Point& start, const G2Point& end);
```

このオブジェクトに始点 start、向きが vector の線分を設定します。
```
void SetLine(const G2Point& start, const G2Vector& vector);
```

11.7.4 判定

このオブジェクトが有効な線分かどうか判定します。
線分長が GeMath::DEps 以上、始点と終点が有効な座標を持てば有効です。
```
bool IsValid() const;
```
戻り値　有効な線分なら true を返します。

ふたつの線分が等価であるか判定します。次の条件が成立すると等価です。
this.GetStartPoint() == line.GetStartPoint() && this.GetEndPoint() == line.GetEndPoint()。

```
bool Equals(const G2Line& line, double tol) const;
```
line　比較する G2Line オブジェクト。
tol　座標を比較するときの許容値。
戻り値　与えられた許容値以内で等価であるとき true を返します。

ふたつの線分が逆向きであるか判定します。次の条件が成立すると逆向きです。
this.GetStartPoint() == lineGetEndPoint() && this.GetEndPoint() == lineGetStartPoint()。
```
bool IsReversed(const G2Line& line, double tol) const;
```
line　比較する G2Line オブジェクト。
tol　座標を比較するときの許容値。
戻り値　与えられた許容値以内で逆向きなら true を返します。

11.7.5 オフセット

この線分のオフセットを計算します。
```
G2Line Offset(int dir, double dist) const;
```
dir　オフセット方向。
G2Math::Left　線分の左側へオフセット。
G2Math::Right　線分の右側へオフセット。
dist　オフセットする距離。

11.7.6 分割／連結

この線分オブジェクトを線分上の点 point の位置で二分割します。
```
int SplitAt(const G2Point& point, double dmin, G2Line* first, G2Line* second) const;
```
この線分オブジェクトを線分のパラメータ lprm の位置で二分割します。
```
int SplitAt(double lprm, double dmin, G2Line* first, G2Line* second) const;
```
dmin　許容値。
first　分割後の前半の線分を格納する G2Line オブジェクトへのポインタ。
second　分割後の後半の線分を格納する G2Line オブジェクトへのポインタ。
戻り値　結果を表す整数。
0　ふたつに分割できた。
1　分割位置が始点と一致。first には何も設定しない。
2　分割位置が終点と一致。second には何も設定しない。
3　分割できない。first、second のどちらも設定しない。
first、second の指すオブジェクトは不要となった時点で delete しなければなりません。

ふたつの線分を連結します。ふたつの線分が同一線上にあり、向きが同じでなければなりません。線分は離れていても、重なりがあってもかまいません。連結可能ならこの線分オブジェクトをふたつの線分の連結線分で置き換えます。
```
bool Merge(const G2Line& second, double tol);
```
second　もうひとつの線分オブジェクト。
tol　許容値。
戻り値　この線分オブジェクトがふたつの線分の連結で置き換えられたとき true を返します。

この線分オブジェクトを等分割します。
```
int Divide(int count, G2Line lines[]) const;
```
count　分割数。2 <= count。
lines　分割してできた線分オブジェクトを格納する配列。lines[count]。
戻り値　計算した線分オブジェクトの数を返します。

11.7.7 最短距離

この線分オブジェクトと幾何図形（G2Point、G2Line、G2Circle、G2Bezier）間の最短距離を計算します。

```
int Measure(int iswt, const G2Point& point, double* dist, G2Point pons[]) const;
int Measure(int iswt, const G2Line& line,   double* dist, G2Point pons[]) const;
int Measure(int iswt, const G2Circle& cir,  double* dist, G2Point pons[]) const;
int Measure(int iswt, const G2Bezier& bzc,  double* dist, G2Point pons[]) const;
```

iswt　選択肢。
- 0　無限長。線分なら直線、円弧なら円として処理します。
- 1　有限長。

dist　最短距離 (0 <= *dist)。

pons　最短点を格納する G2Point オブジェクトの配列。pons[0] はこの線分オブジェクト上の点、pons[1] は引の数幾何図形上の点を格納します。

戻り値　最短距離が求まったとき 0 を返します (0 -)。

11.7.8 点の計算

この線分オブジェクトと曲線 (G2Line、G2Circle、G2Bezie) の交点を計算します。

```
int Intersection(const G2Line& line,  G2Point pins[], double lprms[][2]) const;
int Intersection(const G2Circle& cir, G2Point pins[], double lprms[][2]) const;
int Intersection(const G2Bezier& bzc, G2Point pins[], double lprms[][2]) const;
```

pins　交点を格納する配列。

lprms　交点位置の曲線パラメータを格納する配列。
- lprms[][0]　この線分のパラメータ。
- lprms[][1]　引数の曲線のパラメータ。

戻り値　出力配列に格納した交点の数 (0-9)。

次のメソッドは常に線分の両端点だけを返します。

```
int PointSamplerTol(int mode, double tol, int maxtry, G2PointArray* points) const;
```

points　点を追加する G2PointArray オブジェクトへのポインタ。このオブジェクトに追加できる点数が 2 以上でなければなりません。

戻り値　追加した点の数 (0 か 2)。

引数 mode、tol、maxtry は参照しません。

11.7.9 共通接円弧

この線分オブジェクトと幾何図形（G2Point、G2Line、G2Circle、G2Bezier）の共通接円弧を計算します。円弧の始点はこの線分オブジェクト上にあり、終点は引数の幾何図形上にあります

```
int Fillet(const G2Point& point, double radius, G2CirclePtrArray* pArcs) const;
int Fillet(const G2Line& line,   double radius, G2CirclePtrArray* pArcs) const;
int Fillet(const G2Circle& cir,  double radius, G2CirclePtrArray* pArcs) const;
int Fillet(const G2Bezier& bzc,  double radius, G2CirclePtrArray* pArcs) const;
```

radius　接円の半径 (0 < radius)。

pArcs　円弧を追加する G2CirclePtrArray オブジェクトへのポインタ。

戻り値　追加した円弧の数を返します (0 -)。

11.8 G2Circle クラス

このクラス は円／円弧を表現するクラスです。G2Curve クラスを継承します。以下では基底クラスから継承したメソッドは除き、このクラスで追加するメソッドを主に説明します。

円弧は左回り、右回りかの向きがあります。円弧の向きにより中心角の定義域が異なります。

反時計回り円弧　　0 <= 中心角 <= 2π

時計回り円弧　　-2π <= 中心角 < 0

11.8.1 コンストラクタ

G2Circle();
　中心 (0, 0)、半径 0 の円オブジェクト。
G2Circle(const G2Point& center, double radius);
　中心 center、半径 radius の円オブジェクト。
G2Circle(const G2Point& center, const G2Point& start);
　中心 center、始点 start の円オブジェクト。

11.8.2 Getter

この円の中心点を得ます。
const G2Point& GetCenter() const;
この円の半径を得ます。
double GetRadius() const;

この円の中心角を得ます。
double GetAngleExtent() const;
この円弧の向きを得ます。
int GetDirection() const;
　戻り値　円弧の向きを返します。
　　G2Math::CCw　　反時計回り円弧
　　G2Math::Cw　　時計回り円弧

この円の始点の角度を計算します。
double GetAngleStart() const;
　戻り値　中心点から始点へ引いた半直線の角度を返します。

円始点からの角度を計算します。中心点→始点から中心点→指定した点への角度を計算します。角度は円弧の向きに測ります。円弧が反時計回りなら反時計回りに測定します。円弧が時計回りなら時計回りに測定します。
double AngleAt(const G2Point& point) const;
　point　　角度を測定する点。この円の中心点と等価でないこと。
　戻り値　円始点からの角度を返します。(0 <= angle < 2π)。

この円弧の中点を計算します。
G2Point CalcMidPoint() const;
　戻り値　円弧の中点を返します。

11.8.3 Setter

円／円弧計算メソッドは円の半径が G2Math::DEps 以下になるときはエラーとします。

このオブジェクトに中心、半径の円を設定します。
bool SetCircle(const G2Point& center, double radius);
　center　　中心点。
　radius　　半径。G2Math::DEps < radius。
　戻り値　半径が正しい値なら true を返します。

このオブジェクトに中心、始点の円を設定します。
bool SetCircle(const G2Point& center, const G2Point& start);
　center　　中心点。

start　円の始点。
戻り値　半径が正しい値なら true を返します。

このオブジェクトに始点、通過点、終点の円弧を設定します。この 3 点はひとつの直線上にないことが条件です。例外は始点と終点を等しくしたときで、始点と通過点を直径とする円を計算します。
```
bool SetArc(const G2Point& start, const G2Point& mid, const G2Point& end);
```
start　円弧の始点。
mid　円弧の通過点。
end　円弧の終点。
戻り値　有効な円弧が計算できたら true を返します。

このオブジェクトに中心 center、始点 start、終点 end の円弧を設定します。
```
bool SetArc(const G2Point& center, const G2Point& start, const G2Point& end, int dir);
```
center　中心点。
start　円弧の始点。
end　円弧の終点。
dir　円弧の向き。
　G2Math::CCw　反時計回り円弧
　G2Math::Cw　時計回り円弧
　G2Math::Narrow　劣角を持つ円弧
戻り値　有効な円弧が計算できたら true を返します。

この円弧を円にします。
```
void CloseCircle();
```

この円弧を補円弧にします。補円弧の中心角は（2π - θ、θ は円弧の中心角）、円弧の向きは逆になります。
```
void ToComplement(double tol);
```
tol　許容差。このオブジェクトが円か円弧か判定するのに使います。

11.8.4 判定

このオブジェクトが有効な円弧かどうか判定します。
半径が 0 より大きく、中心点、始点と終点が有効な座標を持てば有効です。
```
bool IsValid() const;
```
戻り値　有効な円弧なら true を返します。

ふたつの円弧が等価であるか判定します。次の条件が成立すると等価です。
this.GetCenter() == arc.GetCenter() &&
this.GetStartPoint() == arc.GetStartPoint() &&
this.GetEndPoint() == arc.GetEndPoint() &&
中心角／円弧の向きが等しい。
```
bool Equals(const G2Circle& arc, double tol) const;
```
arc　比較する G2Circle オブジェクト。
tol　座標、円弧長を比較するときの許容値。
戻り値　与えられた許容値以内で等価であるとき true を返します。

ふたつの円が等価であるか判定します。半径と中心点が等しい等価です。
```
bool EqualCircle(const G2Circle& cir, double tol) const;
```
cir　比較する G2Circle オブジェクト。
tol　座標、円弧長を比較するときの許容値。
戻り値　与えられた許容値以内で等価であるとき true を返します。

この円が反時計回りか判定します。

```
bool IsCCwArc() const;
```
戻り値　この円が反時計回りなら true を返します。

このオブジェクトが完全円か判定します。
```
bool IsCircle(double tol) const;
```
tol　　円弧長を比較するときの許容値。
戻り値　この円弧が許容値以内で完全円なら true を返します。

ふたつの円弧が逆向きであるか判定します。逆向きは円弧の向きだけが反対で、一方の円弧の向きを逆にすれば Equals() が true を返します。
```
bool IsReversed(const G2Circle& arc, double tol) const;
```
arc　　比較する G2Circle オブジェクト。
tol　　座標、円弧長を比較するときの許容値。
戻り値　与えられた許容値以内で逆向きなら true を返します。

11.8.5 オフセット

この円弧のオフセットを計算します。
```
int Offset(int dir, double dist, G2Circle* cir) const;
```
dir　　オフセット方向。
- G2Math::OutSide　　円の外側へオフセット。
- G2Math::InSide　　円の内側へオフセット。

dist　　オフセットする距離。
cir　　オフセットした円弧を格納する G2Circle オブジェクトへのポインタ。
戻り値　結果を示す整数。
- 0　成功
- 1　警告：内側オフセットの距離が半径より大きい。オフセット円の半径は (dist - rad) になり、円弧の向きは逆になります。
- 2　エラー：オフセット円の半径が 0 になるのでオフセットできない。

11.8.6 分割／連結

この円弧オブジェクトを円弧上の点 point の位置で二分割します。
```
int SplitAt(const G2Point& point, double dmin, G2Circle* first, G2Circle* second) const;
```
この円弧オブジェクトを円弧のパラメータ lprm の位置で二分割します。
```
int SplitAt(double lprm, double dmin, G2Circle* first, G2Circle* second) const;
```
dmin　　許容値。
first　　分割後の前半の円弧を格納する G2Circle オブジェクトへのポインタ。
second　分割後の後半の円弧を格納する G2Circle オブジェクトへのポインタ。
戻り値　結果を表す整数。
- 0　　ふたつに分割できた。
- 1　　分割位置が始点と一致。first には何も設定しない。
- 2　　分割位置が終点と一致。second には何も設定しない。
- 3　　分割できない。first、second のどちらも設定しない。

first、second の指すオブジェクトは不要となった時点で delete しなければなりません。

ふたつの円弧を連結します。ふたつの円弧が同一円上にあり、同じ向きで、接続していなければなりません。連結可能ならこの円弧オブジェクトをふたつの円弧の連結線分で置き換えます。
```
bool Merge(const G2Circle& second, double tol)
```
second　もうひとつの円弧オブジェクト。
tol　　許容値。
戻り値　この円弧オブジェクトがふたつの円弧の連結で置き換えられたとき true を返します。

この円弧オブジェクトを等分割します。

```
int Divide(int count, G2Circle arcs[]) const;
```
count　分割数。2 <= count。
lines　分割してできた円弧オブジェクトを格納する配列。lines[count]。
戻り値　計算した円弧オブジェクトの数を返します。

11.8.7 最短距離

この円弧オブジェクトと幾何図形（G2Point、G2Line、G2Circle、G2Bezier）間の最短距離を計算します。

```
int Measure(int iswt, const G2Point& point, double* dist, G2Point pons[]) const;
int Measure(int iswt, const G2Line& line,   double* dist, G2Point pons[]) const;
int Measure(int iswt, const G2Circle& cir,  double* dist, G2Point pons[]) const;
int Measure(int iswt, const G2Bezier& bzc,  double* dist, G2Point pons[]) const;
```
iswt　選択肢。
- 0　無限長。線分なら直線、円弧なら円として処理します。
- 1　有限長。

dist　最短距離 (0 <= *dist)。
pons　最短点を格納する G2Point オブジェクトの配列。pons[0] はこの円弧オブジェクト上の点、pons[1] は引数の幾何図形上の点を格納します。
戻り値　最短距離が求まったとき 0 を返します (0 -)。

11.8.8 点の計算

この円オブジェクトと曲線 (G2Line、G2Circle、G2Bezie) の交点を計算します。

```
int Intersection(const G2Line& line,  G2Point pins[], double lprms[][2]) const;
int Intersection(const G2Circle& cir, G2Point pins[], double lprms[][2]) const;
int Intersection(const G2Bezier& bzc, G2Point pins[], double lprms[][2]) const;
```
pins　交点を格納する配列。
lprms　交点位置の曲線パラメータを格納する配列。
- lprms[][0]　この円弧のパラメータ。
- lprms[][1]　引数の曲線のパラメータ。

戻り値　出力配列に格納した交点の数 (0-9)。

円弧上の点列を結ぶ折れ線と円弧との差が指定した許容差に収まるような点列を計算します。点数は許容差から算出します。

```
int PointSamplerTol(int mode, double tol, int maxtry, G2PointArray* points) const;
```
mode　許容値の種類。
- 1　相対値。(0.0001 <= 弧高 / 弦長 <= 0.25)
- 2　絶対値。弧高

tol　許容値の種類に従った許容値。
maxtry　許容差から算出した点数が G2PointArray オブジェクトに追加できる点数を超えるときの選択。
- 1　算出した点数が追加できる点数を超えるときは計算しない。
- 2 －　算出した点数が追加できる点数を超えるときは、追加できる点数を採用する。この引数の値が 2 ではなく 2 以上となっているのは、G2Bezier クラスのメソッドの引数と合わせるためです。

points　点を追加する G2PointArray オブジェクトへのポインタ。許容差から算出した点数が、このオブジェクトに追加できる点数を超える時は、引数 maxtry の指定に従います。少なくとも 3 点以上追加できなければなりません。
戻り値　追加した点の数 (0 か 3-)。

円に対する点数の見積り。
相対誤差 (0.0001 < 円弧 / 弦長 < 0.25)

```
    tol : 0.0001        n : 7854
    tol : 0.001         n :  785
    tol : 0.01          n :   79
    tol : 0.02          n :   39
    tol : 0.05          n :   16
    tol : 0.1           n :    8
  絶対誤差 (0.00001 < 弧高 / 半径 < 1)
    tol/rad : 0.00001   n :  702
    tol/rad : 0.0001    n :  222
    tol/rad : 0.001     n :   70
    tol/rad : 0.01      n :   22
    tol/rad : 0.02      n :   16
    tol/rad : 0.05      n :   10
    tol/rad : 0.1       n :    7
```

11.8.9 共通接線

この円オブジェクトと幾何図形（G2Point、G2Circle、G2Bezie) の共通接線を計算します。接線の始点はこの円オブジェクト上にあり、終点は引数の幾何図形上にあります。

```
int Tangent(const G2Point& point, G2LinePtrArray* pLines) const;
int Tangent(const G2Circle& cir,  G2LinePtrArray* pLines) const;
int Tangent(const G2Bezier& bzc,  G2LinePtrArray* pLines) const;
```

pLines　線分を追加する G2LinePtrArray オブジェクトへのポインタ。
戻り値　追加した線分の数を返します (0 -)。

11.8.10 共通接円弧

この円オブジェクトと幾何図形（G2Point、G2Line、G2Circle、G2Bezier）の共通接円弧を計算します。円弧の始点はこの円オブジェクト上にあり、終点は引数の幾何図形上にあります。

```
int Fillet(const G2Point& point, double radius, G2CirclePtrArray* pArcs) const;
int Fillet(const G2Line& line,   double radius, G2CirclePtrArray* pArcs) const;
int Fillet(const G2Circle& cir,  double radius, G2CirclePtrArray* pArcs) const;
int Fillet(const G2Bezier& bzc,  double radius, G2CirclePtrArray* pArcs) const;
```

rasius　接円の半径 (0 < radius)。
pArcs　円弧を追加する G2CirclePtrArray オブジェクトへのポインタ。

11.8.11 ３つの図形に接する円

３つの幾何図形に接する円を計算します。

```
static bool TangentCircles3(const G2Geom* geoms[], int maxitr, double deps,
                            const G2Point& center, const G2Point pner[],
                            G2Circle* circle);
```

geoms	G2Geom オブジェクトのポインタ配列。geoms[3]。
maxitr	最大繰返し数（1 <= maxitr)。
deps	許容値。
center	円中心点の近似値。
pner	3 接点の近似値。pner[3]。
circle	接円を格納する G2Circle オブジェクトへのポインタ。
戻り値	接円が求まったとき true を返します。

複数の接円が存在する場合は、与えられた円中心点の近似値 center 付近に中心点を持つ接円を求めます。入力 geoms が３つの円の場合は、3 接点が近似値 pner に最も近い接円を選択します。pner[i] は幾何図形 geoms[i] が円弧のときだけ参照します。

11.9 G2Bezier クラス

このクラスは２次元の 3 次 Bezier 曲線を表現するクラスです。G2Curve クラスを継承します。以下では基底クラスから継承したメソッドは除き、このクラスで追加するメソッドを主に説明します。

11.9.1 コンストラクタ

```
G2Bezier();
```
始点、制御点、終点の全てが (0,0) の Bezier 曲線オブジェクト。
```
G2Bezier(const G2Point& ps, const G2Point& p1, const G2Point& p2, const G2Point& pe);
```
始点 ps、制御点 p1、p2、終点 pe の Bezier 曲線オブジェクト。

11.9.2 Getter

この Bezier 曲線の制御点 1 を得ます。
```
const G2Point& GetCtrlPoint1() const;
```
この Bezier 曲線の制御点 2 を得ます。
```
const G2Point& GetCtrlPoint2() const;
```

11.9.3 Setter

このオブジェクトに Bezier 曲線定義点（始点、制御点、終点）を設定します。
```
void SetBezier(const G2Point& ps, const G2Point& p1, const G2Point& p2, const G2Point& pe);
```
ps	始点。
p1	制御点 1。
p2	制御点 2。
pe	終点。

このオブジェクトの曲線定義点を変更します。
```
void SetPoint(int id, const G2Point& location);
```
id	変更する点を示す整数。
0	始点。
1	制御点 1。
2	制御点 2。
3	終点。
location	新しい位置。

このオブジェクトの曲線定義点を移動します。
```
void TranslatePoint(int id, const G2Point& vector);
```
id	移動する点を示す整数。
0	始点。
1	制御点 1。
2	制御点 2。
3	終点。
vector	移動量を与えます。座標 += 移動量となります。

このオブジェクトの曲線長を再計算します。曲線長は曲線定義点に変更があれば自動的に再計算しますので、通常はこのメソッドを使うことはありません。
```
void RecalcLength();
```

11.9.4 判定

このオブジェクトが有効な Bezier 曲線かどうか判定します。
曲線定義点が有効な座標を持ち、曲線長が 0 より大きければ有効です。
```
bool IsValid() const;
```
戻り値　有効な Bezier 曲線なら true を返します。

ふたつの Bezier 曲線が等価であるか判定します。次の条件が成立すると等価です。
```
this.GetStartPoint() == bzc.GetStartPoint() &&
this.GetCtrlPoint1() == bzc.GetCtrlPoint1() &&
this.GetCtrlPoint2() == bzc.GetCtrlPoint2() &&
this.GetEndPoint() == bzc.GetEndPoint()
```
```
bool Equals(const G2Bezier& bzc, double tol) const;
```
bzc　　比較する G2Bezier オブジェクト。
tol　　座標、曲線長を比較するときの許容値。
戻り値　与えられた許容値以内で等価であるとき true を返します。

ふたつの Bezier 曲線が逆向きであるか判定します。次の条件が成立すると逆向きです。
```
this.GetStartPoint() == bzc.GetEndPoint() &&
this.GetCtrlPoint1() == bzc.GetCtrlPoint2() &&
this.GetCtrlPoint2() == bzc.GetCtrlPoint1() &&
this.GetEndPoint() == bzc.GetStartPoint()
```
```
bool IsReversed(const G2Bezier& bzc, double tol) const;
```
bzc　　比較する G2Bezier オブジェクト。
tol　　座標を比較するときの許容値。
戻り値　与えられた許容値以内で逆向きなら true を返します。

この Bezier 曲線オブジェクトが一直線上にあるか判定します。
```
bool IsLinear(double tol) const;
```
tol　　座標を比較するときの許容値。
戻り値　与えられた許容値以内で一直線上にあるなら true を返します。

11.9.5 オフセット

3 次 Bezier 曲線のオフセットを 3 次 Bezier 曲線で精確に表現することはできません。このメソッドで得るオフセットは近似です。オフセット距離が大きくなるほど真のオフセットとの差異が大きくなります。このメソッドは曲線に変曲点があると、真のオフセットとの差異を少なくするため、変曲点で分割したふたつのオフセット曲線を計算します。

```
int Offset(int dir, double dist, G2Bezier bzcs[]) const;
```
dir　　オフセット方向。
　G2Math::Left　　曲線の右側へオフセット。
　G2Math::Right　　曲線の左側へオフセット。
dist　　オフセットする距離。
bzcs　　オフセットした曲線を格納する G2Bezier オブジェクトの配列。
戻り値　オフセット曲線の数。
　0　オフセットできない。
　1　オフセット曲線はひとつ。
　2　オフセット曲線はふたつ。この曲線には変曲点があり、変曲点で分割してふたつのオフセット曲線を返す。

11.9.6 分割／連結

この Bezier 曲線オブジェクトを Bezier 曲線上の点 point の位置で二分割します。
```
int SplitAt(const G2Point& point, double dmin, G2Bezier* first, G2Bezier* second) const;
```
この Bezier 曲線オブジェクトを Bezier 曲線のパラメータ lprm の位置で二分割します。
```
int SplitAt(double lprm, double dmin, G2Bezier* first, G2Bezier* second) const;
```
dmin　許容値。
first　分割後の前半の Bezier 曲線を格納する G2Bezier オブジェクトへのポインタ。
second　分割後の後半の Bezier 曲線を格納する G2Bezier オブジェクトへのポインタ。
戻り値　結果を表す整数。
- 0　ふたつに分割できた。
- 1　point が始点と一致。first には何も設定しない。
- 2　point が終点と一致。second には何も設定しない。
- 3　分割できない。first、second のどちらにも何も設定しない。

first、second の指すオブジェクトは不要となった時点で delete しなければなりません。

ふたつの Bezier 曲線を連結します。ふたつの Bezier 曲線は接続していなければなりません。連結可能ならこの Bezier 曲線オブジェクトをふたつの Bezier 曲線の連結線分で置き換えます。
```
bool Merge(const G2Bezier& second, double tol);
```
second　もうひとつの Bezier 曲線オブジェクト。
tol　許容値。
戻り値　この Bezier 曲線オブジェクトがふたつの Bezier 曲線の連結で置き換えられたとき true を返します。

11.9.7 最短距離

この Bezier 曲線オブジェクトと幾何図形（G2Point、G2Line、G2Circle、G2Bezier）間の最短距離を計算します。
```
int Measure(int iswt, const G2Point& point, double* dist, G2Point pons[]) const;
int Measure(int iswt, const G2Line& line,   double* dist, G2Point pons[]) const;
int Measure(int iswt, const G2Circle& cir,  double* dist, G2Point pons[]) const;
int Measure(int iswt, const G2Bezier& bzc,  double* dist, G2Point pons[]) const;
```
iswt　選択肢。
- 0　無限長。線分なら直線、円弧なら円として処理します。
- 1　有限長。

dist　最短距離 (0 <= *dist)。
pons　最短点を格納する G2Point オブジェクトの配列。pons[0] はこの Bezier 曲線オブジェクト上の点、pons[1] は引数の幾何図形上の点を格納します。
戻り値　最短距離が求まったとき 0 を返します (0 -)。

11.9.8 点の計算

この Bezier 曲線オブジェクトと曲線 (G2Line、G2Circle、G2Bezier) の交点を計算します。
```
int Intersection(const G2Line& line,  G2Point pins[], double lprms[][2]) const;
int Intersection(const G2Circle& cir, G2Point pins[], double lprms[][2]) const;
int Intersection(const G2Bezier& bzc, G2Point pins[], double lprms[][2]) const;
```
pins　交点を格納する配列。
lprms　交点位置の曲線パラメータを格納する配列。
- lprms[][0]　この Bezier 曲線のパラメータ。
- lprms[][1]　引数の曲線のパラメータ。

戻り値　出力配列に格納した交点の数 (0-9)。

Bezier 曲線上の点列を結ぶ折れ線と Bezier 曲線との差が指定した許容差に収まるような点列を計算します。点数は許容差と曲線の形状で決まります。指定の許容値で計算すると最大点数を超える場合は、許容値を倍に緩めて試みます。それでも点数が多すぎるときはこれを繰り返しますが、繰り返しの打ち切り回数に達したら停止します。最大点数は G2PointArray オブジェクトを使って指定します。この曲線が直線分と等価であると判断したときは曲線の両端点を返します。

int PointSamplerTol(int mode, double tol, int maxtry, G2PointArray* points) const;

mode　許容値の種類。
- 1　相対値。(0.0001 <= 弧高 / 弦長 <= 0.25)
- 2　絶対値。(弧高)

tol　許容値の種類に従った許容値。

maxtry　繰り返しの打ち切り回数 (1 <= maxtry)。繰り返しをさせないなら 1 を渡す。

points　点を追加する G2PointArray オブジェクトへのポインタ。このオブジェクトに追加できる点数が最大点数になる。この値は２以上でなければならない。

戻り値　追加した点の数 (0 -)。

次のメソッドは近似の等間隔点 — ほぼ等間隔な点列を計算します。近似なのは演算時間を短くするためです。

```
int PointSamplerEsp(int count, G2PointArray* points) const;
```

count　計算する点数 (1 -)。
- 1　曲線の中点を計算する。
- 2　曲線の始点、終点を計算する。
- 3 以上　等間隔点列を計算する (曲線の両端点を含む)。

points　点を追加する G2PointArray オブジェクトへのポインタ。このオブジェクトに追加できる点数が count 以上でなければなりません。

戻り値　追加した点の数 (0 - count)。

11.9.9　共通接線

この Bezier 曲線オブジェクトと幾何図形（G2Point、G2Circle、G2Bezier) の共通接線を計算します。接線の始点はこの Bezier 曲線オブジェクト上にあり、終点は引数の幾何図形上にあります。

```
int Tangent(const G2Point& point, G2LinePtrArray* pLines) const;
int Tangent(const G2Circle& cir,  G2LinePtrArray* pLines) const;
int Tangent(const G2Bezier& bzc,  G2LinePtrArray* pLines) const;
```

pLines　線分を追加する G2LinePtrArray オブジェクトへのポインタ。

戻り値　追加した線分の数を返します (0 -)。

11.9.10 共通接円弧

この Bezier 曲線オブジェクトと幾何図形（G2Point、G2Line、G2Circle、G2Bezier）の共通接円弧を計算します。円弧の始点はこの Bezier 曲線オブジェクト上にあり、終点は引数の幾何図形上にあります。

```
int Fillet(const G2Point& point, double radius, G2CirclePtrArray* pArcs) const;
int Fillet(const G2Line& line,   double radius, G2CirclePtrArray* pArcs) const;
int Fillet(const G2Circle& cir,  double radius, G2CirclePtrArray* pArcs) const;
int Fillet(const G2Bezier& bzc,  double radius, G2CirclePtrArray* pArcs) const;
```

radius　接円の半径 (0 < radius)。

pArcs　円弧を追加する G2CirclePtrArray オブジェクトへのポインタ。

戻り値　追加した円弧の数を返します (0 -)。

11.10 G2Matrix クラス

このクラスは２次元の座標変換行列を表現するクラスです。座標変換行列は座標 (x, y) をひとつの直交座標からもうひとつの直交座標に変換します。座標変換は平行移動、スケーリング、回転、反転により構成されます。

このような座標変換は最終行に [0 0 1] を持つ 3 行 × 3 列の行列によって表現できます。この行列は次式に従って（座標に行列を乗算する）座標 (x, y) を座標 (x', y') に変換します。

```
[ x']   [  m00  m01  m02  ] [ x ]   [ m00 * x + m01 * y + m02 ]
[ y'] = [  m10  m11  m12  ] [ y ] = [ m10 * x + m11 * y + m12 ]
[ 1 ]   [   0    0    1   ] [ 1 ]   [             1           ]
```

11.10.1 コンストラクタ

G2Matrix();
　恒等変換行列になります。
G2Matrix(double m00, double m10, double m01, double m11, double m02, double m12);
　行列の全要素を指定するコンストラクタ。

11.10.2 Getter

このオブジェクトの行列要素を得ます。
void GetMatrix(double matrix[]) const;
　matrix　このオブジェクトの行列要素を格納する配列。m00, m10, m01, m11, m02, m12 の順。

このオブジェクトの回転成分を得ます。単一倍率の場合のみ有効。
G2Vector GetVector() const;
　戻り値　ベクトル（m00, m10）を返します。

このオブジェクトの移動成分を得ます。
G2Point GetTranslation() const;
　戻り値　点（m02, m12）を返します。

11.10.3 Setter

このオブジェクトに恒等変換を設定します。
void SetToIdentity();

このオブジェクトの回転変換部分 (c, s) だけを設定します。

```
[   c   -s   m02  ]
[   s    c   m12  ]
[   0    0    1   ]
```

void SetRotation(double theta);
　theta　回転角。
void SetRotation(const G2Vector& vec);
　vec　回転成分。ベクトルの単位ベクトル成分を使います。
void SetRotation(double c, double s);
　c　回転成分 cos(theta)。
　s　回転成分 sin(theta)。

このオブジェクトの移動変換部分 (tx, ty) だけを設定します。

```
[  m00  m01  tx  ]
[  m10  m11  ty  ]
[   0    0    1  ]
```

void SetTranslation(const G2Point& tvec);
　tvec　移動成分を与える点。
void SetTranslation(double tx, double ty);
　tx　移動成分 X。
　ty　移動成分 Y。

このオブジェクトの回転・移動変換を設定します。
```
void SetTransform(double theta, const G2Point& tvec);
```
theta　回転角。
tvec　移動成分を与える点。

このオブジェクトに指定した点を中心とした回転変換を設定します。
```
void SetToRotation(double theta, const G2Point& center);
```
theta　回転角。
center　回転の中心。
```
void SetToRotation(double c, double s, const G2Point& center);
```
c　回転成分 cos(theta)。
s　回転成分 sin(theta)。
center　回転の中心。

このオブジェクトに指定した点を中心としたスケーリング変換を設定します。
```
void SetToScale(const G2Point& center, double scale);
```
center　スケーリングの中心。
scale　倍率。
```
void SetToScale(const G2Point& center, const double scales[]);
```
center　スケーリングの中心。
scales　２つの倍率。scale[0] != scale[1]。

このオブジェクトに反転変換を設定します。
```
void SetToMirror(const G2Vector& vec, const G2Point& po);
```
vec　反転軸の向き。
po　反転軸の始点。

このオブジェクトに行列の全要素を設定します。
```
void SetMatrix(double m00, double m10, double m01, double m11, double m02, double m12);
```

11.10.4 座標変換

点を座標変換します。
```
void Transform(G2Point* point) const;
```
point　入力点かつ出力点。

点を座標変換します。
```
void Transform(const G2Point& srcPnt, G2Point* dstPnt) const;
```
srcPnt　入力点。
dstPnt　出力点。

点列を座標変換します。
```
void Transform(G2Point points[], int offset, int count) const;
```
points　入力点列かつ出力点列。
offset　点列の開始インデックス (0 -)。
count　変換する点数。
points[offset] から points[offset + count - 1] までを座標変換します。

点列を座標変換します。
```
void Transform(const G2Point srcPts[], int srcOff, G2Point dstPts[], int dstOff, int count) const;
```
srcPts　入力点列。
srcOff　入力点列の開始インデックス (0 -)。
dstPts　出力点列。
dstOff　出力点列の開始インデックス (0 -)。

count　　変換する点数。
srcPts[srcOff] から srcPts[srcOff + count - 1] までを座標変換し、dstPts[dstOff] から dstPts[dstOff + count - 1] に格納します。

点列を座標変換します。
```
void Transform(const double srcPts[], int srcOff, double dstPts[], int dstOff, int numPts) const;
```
srcPts　　入力点列。2 次元座標（x0,y0,x1,y1, ...）を持つ double 配列。
srcOff　　入力点列の開始インデックス (0 -)。
dstPts　　出力点列。2 次元座標（x0,y0,x1,y1,...）を格納する double 配列。
dstOff　　出力点列の開始インデックス (0 -)。
numPts　　変換する点数。
srcPts[srcOff] から srcPts[srcOff + 2*numPts - 1] までを座標変換し、dstPts[dstOff] から dstPts[dstOff + 2*numPts - 1] に格納します。

ベクトルの座標変換は移動変換部分は無視します。
```
void Transform(G2Vector* vector) const;
```
vector　　入力ベクトルかつ出力ベクトル。
```
void Transform(const G2Vector& srcVec, G2Vector* dstVec) const;
```
srcVec　　入力ベクトル。
dstVec　　出力ベクトル。

11.10.5 逆座標変換

点を逆座標変換します。
```
void InverseTransform(G2Point* point) const;
```
point　　入力点かつ出力点。

点を逆座標変換します。
```
void InverseTransform(const G2Point& srcPnt, G2Point* dstPnt) const;
```
srcPnt　　入力点。
dstPnt　　出力点。

点列を逆座標変換します。
```
void InverseTransform(G2Point points[], int offset, int count) const;
```
points　　入力点列かつ出力点列。
offset　　点列の開始インデックス (0 -)。
count　　変換する点数。
points[offset] から points[offset + count - 1] までを座標変換します。

点列を逆座標変換します。
```
void InverseTransform(const G2Point srcPts[], int srcOff, G2Point dstPts[], int dstOff, int count)
    const;
```
srcPts　　入力点列。
srcOff　　入力点列の開始インデックス (0 -)。
dstPts　　出力点列。
dstOff　　出力点列の開始インデックス (0 -)。
count　　変換する点数。
srcPts[srcOff] から srcPts[srcOff + count - 1] までを座標変換し、dstPts[dstOff] から dstPts[dstOff + count - 1] に格納します。

点列を逆座標変換します。
```
void InverseTransform(const double srcPts[], int srcOff, double dstPts[], int dstOff, int numPts) const;
```
srcPts　　入力点列。2 次元座標（x0,y0,x1,y1, ...）を持つ double 配列。
srcOff　　入力点列の開始インデックス (0 -)。

dstPts　　出力点列。2 次元座標（x0,y0,x1,y1,...）を格納する double 配列。
dstOff　　出力点列の開始インデックス (0 -)。
numPts　　変換する点数。
srcPts[srcOff] から srcPts[srcOff + 2*numPts - 1] までを座標変換し、dstPts[dstOff] から dstPts[dstOff + 2*numPts - 1] に格納します。

ベクトルの逆座標変換は移動変換部分は無視します。
```
void InverseTransform(G2Vector* vector) const;
```
vector　　入力ベクトルかつ出力ベクトル。
```
void InverseTransform(const G2Vector& srcVec, G2Vector* dstVec) const;
```
srcVec　　入力ベクトル。
dstVec　　出力ベクトル。

11.10.6 演算子

行列の積を計算する * 演算子と =* 演算子 があります。* 演算子は行列の積を新しい G2Matrix オブジェクトで返します。これに対して =* 演算子 は演算子を呼び出したオブジェクトを計算結果で書き換えます。

この行列オブジェクトに引数の行列を連結した行列オブジェクトを計算します (this * rotation)。
```
G2Matrix  operator*(const G2Matrix& matrix) const;
```
matrix　　G2Matrix オブジェクトの参照。
戻り値　　この行列オブジェクトに引数の行列を連結した行列オブジェクトを返します。

この行列オブジェクトに引数の行列を連結します。(this = this * rotation)。
```
G2Matrix& operator*=(const G2Matrix& matrix);
```
matrix　　G2Matrix オブジェクトの参照。
戻り値　　この行列オブジェクトの参照を返します。

11.10.7 逆変換行列

このオブジェクトの逆変換をする行列オブジェクトを得ます。
```
G2Matrix CreateInverse() const;
```
戻り値　　この行列の逆変換行列オブジェクトを返します。

11.11 G2PointArray クラス

このクラス は可変長の 2 次元点列 (G2Point オブジェクト) を表現するクラスです。多角形としても使います。

11.11.1 コンストラクタ

```
G2PointArray(size_t size = 256);
```
配列の最大サイズ size を指定します。この配列は空です。

11.11.2 最大サイズ

配列の最大サイズを変更します。点オブジェクトを追加した後でもこのメソッドを使って最大サイズを変更できます。ただし実際に格納されている点数はかわりません。実際に格納されている点数より小さな数を設定しても点数は減りません。
```
void SetMaxCount(size_t size);
```
size　　配列の最大サイズ（点数）。

11.11.3 点の取得

この配列が空か判定します。
```
bool IsEmpty() const;
```
この配列が空のとき true を返します。

この配列の点数が上限に達しているか判定します。
```
bool IsFull() const;
```
戻り値　この配列の点数が最大サイズ以上のとき true を返します。

この配列の点数を得ます。
```
int  GetCount() const;
```
戻り値　この配列の点数を返します。

この配列に追加できる点数を得ます。
```
int  GetRemainder() const const;
```
戻り値　この配列に追加できる点数を返します。

この配列の指定した位置の点を得ます。
```
const G2Point& GetAt(int index) const;
```
index　　点の位置を示すインデックス。0 <= index < GetCount()。
戻り値　index の位置の G2Point オブジェクトを返します。インデックスが範囲外のときは G2Point::ORIGIN を返します。

この配列の最後の点を得ます。
```
const G2Point& GetLast() const;
```
戻り値　最後の G2Point オブジェクトを返します。配列が空の時はのときは G2Point::ORIGIN を返します。

この配列の指定した範囲の点列を得ます。
```
int GetPoints(int fromIndex, int toIndex, G2Point points[]) const;
```
fromIndex　開始インデックス。0 <= fromIndex < GetCount()。
toIndex　　終了インデックス。fromIndex <= toIndex。終了インデックスが配列の最後を越える時は配列の最後までとします。
points　　点をコピイする配列。
戻り値　コピイした点数を返します。

G2Point オブジェクト配列への const ポインタを得ます。
```
const G2Point* GetPointer() const;
```
戻り値　G2Point 配列へのポインタを返します。配列が空のときは null ポインタを返します。
このメソッドは関数の引数が G2Point オブジェクト配列の場合に使うためにあります。

11.11.4 点の追加、削除

点をこの配列の最後に追加します。
```
bool Add(const G2Point& point);
```
point　　G2Point オブジェクト。
戻り値　追加したとき true を返します。

点列をこの配列の最後に追加します。全部を追加すると点数が上限に達するときは上限を超える分は追加できません。
```
bool Append(const G2Point points[], int count);
```
points　　追加する点の配列。
count　　追加する点数。

戻り値　点を追加したとき true を返します。

点列をこの配列の最後に追加します。全部を追加すると点数が上限に達するときは上限を超える分は追加できません。
```
bool Append(const G2PointArray& srcArray);
```
srcArray 追加する点列を持つ G2PointArray オブジェクト。
戻り値　点を追加したとき true を返します。

この配列の指定した位置の点を変更します。
```
void SetAt(int index, const G2Point& point);
```
index　位置を示すインデックス。0 <= index < GetCount()。
point　設定する点。

この配列に点を挿入します。インデックスで示された位置とそれに以降にある点を後方に移動し、インデックスの位置に点を追加します。インデックス 0 は点を先頭に挿入します。
```
bool InsertAt(int index, const G2Point& point);
```
index　点の挿入位置を示すインデックス。0 <= index <= GetCount()。
point　挿入する点。
戻り値　点を挿入したら true を返します。

この配列を空にします。
```
void RemoveAll();
```

この配列の最後の点を削除します。
```
void RemoveLast();
```

この配列の指定した位置の点を削除します。
```
void RemoveAt(int index);
```
index　削除する点の位置を示すインデックス。0 <= index < GetCount()。

この配列の指定した範囲の点列を削除します。
```
void Remove(int fromIndex, int toIndex);
```
fromIndex　開始インデックス。0 <= fromIndex < GetCount()。
toIndex　終了インデックス。fromIndex <= toIndex。終了インデックスが配列の最後を越える時は配列の最後までとします。

11.11.5 判定

この配列が指定した点があるか調べます。
```
int Find(const G2Point& point, double tol) const;
```
point　探索する点。
tol　点を比較するときの許容差。
戻り値　この配列のインデックスを返します。この点がないときは -1 を返します。

この配列の点の中で指定した点に最も近い点を探します。
```
int ChoosePoint(const G2Point& point) const;
```
point　基準点。
戻り値　この配列のインデックスを返します。配列が空なら -1 を返します。

指定した点がこの配列の最後の点と等しいか判定します。
```
bool EqualsLastPoint(const G2Point& point, double tol) const;
```
point　比較する点。
tol　点を比較するときの許容差。
戻り値　等しいと true を返します。

この配列が表現する多角形が閉じているか判定します。
```
bool IsClosed(double tol) const;
```
tol　　点を比較するときの許容差。
戻り値　点数が 2 以上で、最初と最後の点が等しいと true を返します。

この配列の全ての点がひとつの直線上にあるか判定します。
```
bool IsLinear(double tol) const;
```
tol　　点を比較するときの許容差。
戻り値　全ての点が一直線上にあると true を返します。点数が 2 未満または全ての点が 1 点に集中している場合も true を返します。

この配列に重複点があるか判定します。
```
bool HasDuplication(double tol, bool full = false) const;
```
tol　　点を比較するときの許容差。
full　　比較方法
　false　　配列のインデックスが隣同士の比較。
　true　　全点を比較。
戻り値　重複点があると true を返します。

この配列が表現する単純多角形が反時計回りか判定します。
```
bool IsCCwPloygon() const;
```
戻り値　反時計回りだと true を返します。

この配列の点列を含む最小矩形を得ます。
```
G2Rect GetBounds() const;
```
戻り値　配列が空なら Invalid な G2Rect オブジェクトを返します。

指定した点が、この配列が表現する単純多角形の内部にあるか判定します。この単純多角形に重複点が無いこと、始点と終点を重複させないこと、点数が 3 以上です。
```
int Contains(const G2Point& point) const;
```
point　　判定する点。
戻り値　結果
　G2Math::OutSide　　多角形の内側
　G2Math::OnCurve　　多角形の辺上
　G2Math::OutSide　　多角形の外側

指定した点が、単純多角形の内部にあるか判定します。
```
static int Contains(const G2Point ply[], int num, const G2Rect& box, const G2Point& point);
```
ply　　単純多角形。重複点が無いこと、始点と終点を重複させないこと。
num　　多角形の点数 (3 <= num)。
box　　単純多角形を包む最小矩形。
point　　判定する点。
戻り値　結果
　G2Math::OutSide　　多角形の内側
　G2Math::OnCurve　　多角形の辺上
　G2Math::OutSide　　多角形の外側

２つの単純多角形の包含関係を調べます。
```
static int Contains(const G2Point ply1[], int n1, const G2Rect& box1,
                    const G2Point ply2[], int n2, const G2Rect& box2);
```
ply1　　単純多角形。重複点が無いこと、始点と終点を重複させないこと。
n1　　多角形の点数。
box1　　単純多角形を包む最小矩形。
ply2　　単純多角形。重複点が無いこと、始点と終点を重複させないこと。
n2　　多角形の点数。

box2　単純多角形を包む最小矩形。
戻り値　結果
0　２つの多角形は離れている。
1　２つの多角形は重なる部分を持つ。
2　多角形 ply2 は 多角形 ply1 を内部に含む。
3　多角形 ply1 は 多角形 ply2 を内部に含む。
点数 n1 または n2 が２以下のときは多角形を構成しないので多角形判定は行なわず、最小矩形だけで判定します。

11.11.6 操作

この配列から重複点を除去します。
```
void PurgeDuplication(double tol, bool full = false);
```
tol　点を比較するときの許容差。
full　比較方法
false　隣同士の比較。
true　全点を比較。

この配列の要素の順序を逆にします。
```
void Reverse(int fromIndex, int toIndex);
```
fromIndex　開始インデックス。0 <= fromIndex < GetCount()。
toIndex　終了インデックス。fromIndex <= toIndex。終了インデックスが配列の最後を越える時は配列の最後までとします。

インデックス first が先頭になるように配列要素を巡回します。
前 : [0] ... [first-1][first] ... [n-1]
後 : [first] ... [n-1][0] ... [first-1]
```
void RotateLeft(int first);
```
first　配列の先頭にする点のインデックス。1 <= fromIndex < GetCount()。

この配列の要素を交換します。
```
void Swap(int index1, int index2);
```
index1　配列のインデックス。1 <= index1 < GetCount()。
index2　配列のインデックス。0 <= index2 < GetCount()。

11.12 G2PtrArray クラステンプレート

このクラステンプレート は幾何図形オブジェクトを格納するコンテナです。
G2CurveArray、G2LinePtrArray、G2CirclePtrArray は G2PtrArray のテンプレートクラスです。

11.12.1 コンストラクタ

```
G2PtrArray(size_t size = 2048);
```
配列の最大サイズ size を指定します。この配列は空です。

11.12.2 メソッド

配列の最大サイズを変更します。要素を追加した後でもこのメソッドを使って最大サイズを変更できます。ただし実際に格納されている要素数はかわりません。実際に格納されている要素数より小さな数を設定しても要素数は減りません。
```
void SetMaxCount(size_t size);
```
size　配列の最大サイズ。

この配列が空か判定します。
```
bool IsEmpty() const;
```
戻り値　この配列が空のとき true を返します。

この配列の点数が上限に達しているか判定します。
```
bool IsFull() const;
```
戻り値　この配列の要素数が最大サイズ以上のとき true を返します。

この配列の要素数を得ます。
```
int  GetCount() const;
```
戻り値　この配列の要素数を返します。

この配列に追加できる要素数を得ます。
```
int  GetRemainder() const;
```
戻り値　この配列に追加できる要素数を返します。

この配列の指定した位置の要素を得ます。
```
const T* GetAt(int index) const;
```
index　　要素の位置を示すインデックス。0 <= index < GetCount()。
戻り値　インデックスの位置の const ポインタを返します。インデックスが範囲外のときは null ポインタを返します。このオブジェクトは G2PtrArray オブジェクトが管理していますので delete してはなりません。
```
T* GetAt(int index);
```
index　　要素の位置を示すインデックス。0 <= index < GetCount()。
戻り値　インデックスの位置のポインタを返します。インデックスが範囲外のときは null ポインタを返します。このオブジェクトは G2PtrArray オブジェクトが管理していますので delete してはなりません。

この配列の最後の要素を得ます。
```
const T* GetLast() const;
```
戻り値　最後の要素の const ポインタを返します。配列が空なら null ポインタを返します。このオブジェクトは G2PtrArray オブジェクトが管理していますので delete してはなりません。

オブジェクトをこの配列の最後に追加します。
```
bool Add(const T& obj);
```
obj　　追加するオブジェクトの参照。
戻り値　この配列にオブジェクトを追加したとき true を返します。
```
bool Add(T* pPtr);
```
pPtr　　追加するオブジェクトへのポインタ。このポインタをそのまま配列に保持するので、このオブジェクトを delete してはなりません。
戻り値　この配列にポインタを追加したとき true を返します。

オブジェクトの配列をこの配列の最後に追加します。全部の要素を追加すると要素数が上限に達するときは上限を超える分は追加できません。
```
bool Append(G2PtrArray<T>* pSrcArray);
```
pSrcArray　このオブジェクトが含む要素をこの配列の最後に追加します。追加した要素はこのオブジェクトから削除されます。
戻り値　　この配列が上限に達していて追加できないとき false を返します。

この配列の指定した位置の要素を削除します。
```
void RemoveAt(int index);
```
index　　削除する要素の位置を示すインデックス。0 <= index < GetCount()。

この配列の最後のの要素を削除します。
```
void RemoveLast();
```

配列を空にします。
```
void RemoveAll();
```

インデックス first が先頭になるように配列要素を巡回します。
```
   前 : [0] ... [first-1][first] ... [n-1]
   後 : [first] ... [n-1][0] ... [first-1]
void RotateLeft(int first);
```
first　　配列の先頭にする点のインデックス。1 <= fromIndex < GetCount()。

この配列の要素を交換します。
```
void Swap(int index1, int index2);
```
index1　　配列のインデックス。1 <= index1 < GetCount()。
index2　　配列のインデックス。0 <= index2 < GetCount()。

11.13 G2Conic クラス

このクラスは円錐曲線（二次曲線）を表現します。このクラスは円錐曲線を計算するのに使用します。円錐曲線をアイテムにするときはこれを近似するスプラインに変換します。AwSplineCreator::AddConic() メソッドを参照ください。

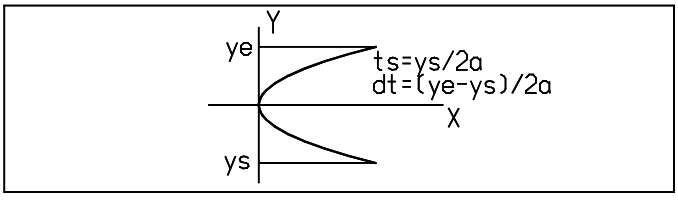
円錐曲線には次の 3 種類の曲線があります。
G2Conic::T_PARABOLA　: 放物線
G2Conic::T_ELLIPSE　　: 楕円
G2Conic::T_HYPERBOLA : 双曲線
円錐曲線 をパラメータ表現した場合にパラメータは曲線上のある点を特定します。ここでのパラメータ (t) は以下のように定めます。

放物線
点 (Po) を原点、ベクトル Uv を正の X 軸とする座標系での曲線の方程式
y^2 = 4ax

$$\begin{cases} x : a*t^2 \\ y = 2*a*t \end{cases}$$

Po　　頂点
a, b　　放物線の方程式の定数 (a > 0, b = 0)
Uv　　頂点から焦点の向き
ts　　始点におけるパラメータ
dt　　終点におけるパラメータ － 始点におけるパラメータ

放物線



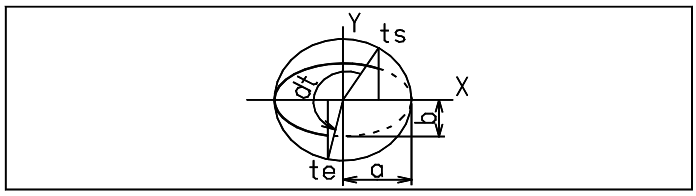
楕円
点 (Po) を原点、ベクトル Uv を正の X 軸とする座標系での曲線の方程式

x^2/a^2 + y^2/b^2 = 1

$$\begin{cases} x = a*\cos(t) \\ y = b*\sin(t) \end{cases} \qquad 0 <= t <= 2\pi$$

Po　　中心点
a, b　　楕円の方程式の定数（長軸と短軸の半径）(a,b > 0)
Uv　　正の長軸の向き
ts　　始点の離心角
dt　　終点の離心角 － 始点の離心角

楕円

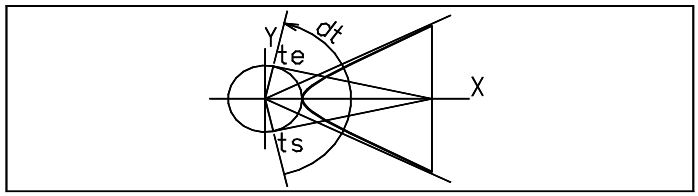


双曲線
点 (Po) を原点、ベクトル Uv を正の X 軸とする座標系での曲線の方程式
x^2/a^2 - y^2/b^2 = 1

$$\begin{cases} x = a*\sec(t) \\ y = b*\tan(t) \end{cases} \qquad -\pi/2 < t < \pi/2$$

t = -π/2、π/2 付近では x,y の値が極めて大きくなります。
Po　　中心点
a, b　　双曲線の方程式の定数 (a = 補助円の半径)(a,b > 0)
Uv　　中心点から頂点の向き
ts　　始点の離心角
dt　　終点の離心角 － 始点の離心角

双曲線



11.13.1 コンストラクタ

G2Conic();
　IsValid() メソッドが false を返すオブジェクトです。

11.13.2 Getter

このオブジェクトが有効な曲線かどうか判定します。
bool IsValid() const;
　戻り値　有効な円錐曲線のとき true を返します。

このオブジェクトの円錐曲線の種類を得ます。

```
int  GetType() const;
```
戻り値　このオブジェクトの円錐曲線の種類を返します。

このオブジェクトの円錐曲線のパラメータを得ます。
```
double GetParamA() const;
```
戻り値　このオブジェクトの円錐曲線のパラメータ a を返します。
```
double GetParamB() const;
```
戻り値　このオブジェクトの円錐曲線のパラメータ b を返します。
```
G2Point GetCenter() const;
```
戻り値　このオブジェクトの円錐曲線の中心を返します。
```
G2Vector GetMajorAxis() const;
```
戻り値　このオブジェクトの円錐曲線の主軸を返します。

11.13.3 Setter

このオブジェクトに放物線を設定します。
```
bool SetParabola(const G2Point points[], int count);
```
points　放物線を決定する点。
- points[0]　放物線の頂点。
- points[1]　放物線の焦点。
- points[2]　弧の始点、終点を決める点。省略すると弧は主軸 100 の位置で切断します。

count　入力点の数 (2-3)。
戻り値　放物線が計算できたら true を返します。

このオブジェクトに放物線を設定します。
```
bool SetParabola(const G2Point points[], const G2Vector& axis);
```
points　放物線を決定する点。
- points[0]　弧の始点。
- points[1]　弧の通過点。
- points[2]　弧の終点。

axis　主軸の向き。ベクトルの大きさは関係ありません。
戻り値　放物線が計算できたら true を返します。

このオブジェクトに楕円を設定します。
```
bool SetEllipse(double major, double minor, const G2Point& center, const G2Vector& axis);
```
major　長半径 (0 < major)。
minor　短半径 (0 < minor <= major)。
center　中心点。
axis　主軸の向き。ベクトルの大きさは関係ありません。
戻り値　楕円が計算できたら true を返します。

このオブジェクトに楕円を設定します。
```
bool SetEllipse(const G2Point points[]);
```
points　楕円の長軸 (主軸) と短軸上の 4 点。長軸と短軸は直交していなければなりません。
- points[0]-[2]　**長軸**。
- points[1]-[3]　**短軸**。

戻り値　楕円が計算できたら true を返します。

このオブジェクトに双曲線を設定します。
```
bool SetHyperbola(double major, double minor, const G2Point& vertex, const G2Vector& axis);
```
major　長半径 (0 < major)。
minor　短半径 (0 < minor <= major)。
vertex　頂点。
axis　主軸の向き。ベクトルの大きさは関係ありません。
戻り値　双曲線が計算できたら true を返します。

11.14関数

● 関数一覧

gmuclip2	**線分を凸多角形領域でクリップする**
gmuclip3	**多角形を多角形領域でクリップする**
xquadeq	**２次方程式を解く**

11.14.1 gmuclip2() 関数

線分を凸多角形領域でクリップする。

【呼出し形式】

```
int gmuclip2(int nvtx, const DPOINT cbry[], const DPOINT vnrm[],
             const DRECT* cext, int iswt, const DLINE* lorg,
             int* key, DLINE* lout)
```

【入力引数】

nvtx	凸多角形の頂点の数（ = 辺の数 ）
cbry	凸多角形の頂点の座標の並び
vnrm	凸多角形の辺の法線ベクトルの並び。 vnrm[j] は 辺 cbry[j]->cbry[j+1] の法線ベクトル。 j = nvtx - 1 のとき辺は cbry[nvtx - 1] -> cbry[0]。 (j = 0 から nvtx - 1)
cext	凸多角形を含む最小矩形
iswt	0
lorg	クリップする線分

【出力引数】

key	結果 -1 ：線分は矩形領域の外にある。 0 ：線分は矩形領域の内にある。 1 ：線分は矩形領域でクリップされた。
lout	クリップされた線分。lorg とおなじ変数でもよい。

【戻り値】

lout	クリップされた線分の数 (0、1)

11.14.2 gmuclip3() 関数

多角形を多角形領域でクリップする。

【呼出し形式】

```
int gmuclip3(int nw, const DPOINT pw[], int np, DPOINT pp[], int maxpnt, DPOINT pq[])
```

【入力引数】

nw	クリップ多角形の頂点の数 (= 辺の数)
pw	クリップ多角形の頂点の座標の並び。 多角形の頂点列は反時計回りであること。
np	クリップしたい多角形の頂点の数（ = 辺の数 ）
pp	クリップしたい多角形の頂点の座標の並び。 多角形の頂点列は反時計回りであること。
maxpnt	配列 pp, pq の長さ。 クリップされた多角形の頂点数は、もとの多角形の頂点数より多くなることもある。た

とえば、正方形のウインドウで菱形をクリップした結果は八角形となる。配列 pp, pq の大きさはこれを考慮すること。

pq　　中間結果の格納に使用する。

【出力引数】

pp　　クリップされた多角形の頂点の座標の並び

【戻り値】

クリップされた多角形の頂点の数または辺の数
クリップしたい多角形が、クリップ多角形と交差していないときは 0。

11.14.3 xquadeq() 関数

２次方程式を解く。

【呼出し形式】

```
int xquadeq(const double c[], double t[])
```

【入力引数】

c　　係数の並び
f(t) = a*t^2 + b*t + c = 0
c[0] = a, c[1] = b and c[2] = c

【出力引数】

t　　根

【戻り値】

根の数 (0-2)

第 12 章　図形アイテム

この章は図形アイテムを扱うクラスを説明します。図形アイテムとは以下のアイテムをいいます。

AwItem::POINT	点アイテム
AwItem::LINE	線分アイテム
AwItem::CIRCLE	円弧アイテム
AwItem::SPLINE	スプラインアイテム
AwItem::STRING	ストリングアイテム

線分、円弧、スプライン、ストリングアイテムを「曲線アイテム」と総称します。曲線アイテムは曲線の始点のサブレコードとそれに続く１つ以上の曲線サブレコードからなっています。これらの並びの間に曲線以外のサブレコードが入ってはなりません。線分アイテムは複数の線分サブレコードだけを含めることができ、それらは矛盾なく１つの直線を構成しなければなりません。つまり全ての点が一直線上にあり、始点から終点にむかって順番になければなりません。円弧アイテム、スプラインアイテムも同様に矛盾のないものでなければなりません。ストリングアイテムだけは各種の曲線サブレコードを含むことができます。

12.1 AwCurveOperator クラス

このクラスは曲線アイテムに対して次の計算をします。

- アイテムの曲線長を計算する
- アイテム上の点を計算する
- アイテムへの投影点を計算する
- 二つアイテムの交点を計算する
- 二つアイテムの最短距離を計算する
- 二つアイテムの共通接線を計算する
- 二つアイテムの共通接円弧を計算する

投影点、交点、共通接線、共通接円では、曲線アイテムのピック位置の幾何図形に対して計算を行います。こうすることでピック位置に最も近い解を得ることができます。これで解が求まらなければ、ピック位置の近傍を探索して解を求めます。これはスプラインだけに適用します。線分アイテムは複数の線分サブレコードを持っていたとしても、どれも同一直線上にあるので（線分アイテムの制約による）近傍を探索する意味がありません。円弧アイテムも同様です。ストリングアイテムは事情が違いますが適用しません。スプラインのように滑らかでないことが主な理由です。メソッドにアイテムのピック情報を渡すには後述する AwPickInfo クラスを使います。
スプラインの曲線中間部の非表示部分は計算対象になります。スプラインの両端のトリム部分は計算対象ではありません。

12.1.1 コンストラクタ

```
AwCurveOperator(const AwGeomParam& geomParam);
```

AwGeomParam オブジェクトの参照を渡します。

12.1.2 曲線長

曲線アイテムの長さを得ます。

```
double GetLength(const AwItem& item,
                 int fromIndex = 0, int toInde = xAwItem::MAX_INDEX) const;
```

item	曲線アイテムの参照。
fromIndex	開始サブレコードインデックス。省略時は先頭から。
toIndex	終了サブレコードインデックス。省略時は最後まで。
戻り値	曲線長。曲線中間部の非表示部分の長さも含めます。トリムしたスプラインではトリム部分の長さは含めません。

12.1.3 曲線上の点

曲線アイテムの定義点列を得ます。定義点列は始点とそれに続く曲線の点の並びです。

線分	**[終点]+**
円弧	**[中間点, 終点]+**
Bezier 曲線	**[終点]+**

[]+ は繰返しを表す記号です。

```
bool GetDefinitionPoints(const AwItem& item, G2PointArray* pArray, bool bTrim = true) const;
```

item	曲線アイテムの参照。
pArray	定義点を追加する G2PointArray オブジェクトへのポインタ。
bTrim	スプラインのトリム部分を含めないとき true を設定します。
戻り値	点数が配列の上限を超えたときは false を返します。

曲線アイテムの両端点を得ます。

```
int GetEndPoints(const AwItem& item, G2Point points[]) const;
```

item	曲線アイテムの参照。
points	始終点を格納する配列。
戻り値	始終点の数（点アイテムは 1、曲線アイテムは 2。それ以外はエラー）。

曲線アイテムの両端点から離れた点を計算します。

```
int CalcEndPoints(const AwItem& item, double lprm, G2Point points[]) const;
```

item	曲線アイテムの参照。
lprm	点の位置を指定する曲線パラメータ（始点および終点からの曲線長）。負の値は曲線延長上の点を計算しますが。スプラインの延長は不定です。
points	点を格納する配列。
戻り値	点の数 (=2、それ以外はエラー)。

曲線アイテムの等分割点を計算します。

```
bool CalcPoints(const AwItem& item, int count, G2PointArray* pArray) const;
```

item	曲線アイテムの参照。
count	計算する点数。
1	中点。
2	始終点。
3	始点、中点、終点。
4	始点、等分割点 (2 点)、終点。
5 -	以下同様。
pArray	点を追加する G2PointArray オブジェクトへのポインタ。
戻り値	点数が配列の上限を超えたときは false を返します。

曲線アイテム上に等間隔な点列を計算します。最大点数に達するか、曲線の終端に達したら終了します。

```
bool CalcPoints(const AwItem& item, double interval, bool backward, int maxpnt,
                G2PointArray* pArray) const;
```

item	曲線アイテムの参照。
interval	点の間隔 (曲線長。0 < interval)。
backward	曲線終点から始点に向かう点列を計算するするとき true にします。
maxpnt	計算する最大点数 (1-)。
pArray	点を追加する G2PointArray オブジェクトへのポインタ。
戻り値	点数が配列の上限を超えたときは false を返します。

曲線アイテム上に等間隔な点列を計算します。最大点数に達するか、曲線の終端に達した時点で終了します。

```
bool CalcPoints(const AwItem& item, double start, double interval, bool backward, int maxpnt,
                G2PointArray* pArray) const;
```

item	曲線アイテムの参照。
start	開始位置 (曲線長。0 < start)。
interval	点の間隔 (曲線長。0 < interval)。
backward	曲線終点から始点に向かう点列を計算するするとき true にします。
maxpnt	計算する最大点数 (1-)。
pArray	点を追加する G2PointArray オブジェクトへのポインタ。
戻り値	点数が配列の上限を超えたときは false を返します。

12.1.4 投影点

点を曲線アイテム上に正投影した点を計算します。曲線アイテムのピック位置の幾何図形に対して投影点を計算します。

```
bool CalcProjections2(const AwPickInfo& info, const G2Point& base, G2PointArray* pArray) const;
```

info	曲線アイテムのピック情報。アイテムの曲線部分（LINE, CIRCLE または BEZIER のいずれか）を指していなければなりません。
base	曲線アイテムに投影する点。
pArray	点を追加する G2PointArray オブジェクトへのポインタ。
戻り値	点数が配列の上限を超えたときは false を返します。

例) 点 (20, 10) を曲線アイテムに投影。

```
// Pick curve item.
const AwItem* pItem = PickItem(0, *token, 1);
if (pItem == NULL || ! pItem->IsCurve())
  return;
const IDENTINFO* idtinfo = IdentInfo();
AwPickInfo info;
if (! info.SetInfo(*pItem, idtinfo->index, idtinfo->styp, idtinfo->geom, token->pnt))
  return;
// Calculate projection.
G2PointArray pnts(16);
AwCureveOperator operator(*(pModel->GetGeomParam()));
if (! operator.CalcProjection2(info, G2Point(20.0, 10.0), &pnts))
  return;
```

12.1.5 交点

二つの曲線アイテムの交点を計算します。曲線アイテムのピック位置の幾何図形に対して交点を計算します。

```
bool CalcIntersections2(const AwPickInfo infos[], G2PointArray* pArray) const;
```

infos	二つの曲線アイテムのピック情報。アイテムの曲線部分（LINE, CIRCLE または BEZIER のいずれか）を指していなければなりません。
pArray	点を追加する G2PointArray オブジェクトへのポインタ。
戻り値	点数が配列の上限を超えたときは false を返します。

例) 曲線アイテムの交点。

```
// Pick two curve items.
AwPickInfo infos[2];
for (int j = 0; j < 2; ++j) {
  // ここで token を適切に設定するとする。
  const AwItem* pItem = PickItem(0, token, 1);
  if (pItem == NULL || ! pItem->IsCurve())
    return;
  if (j == 1 && pItem->GetItemId() == infos[0].GetItemId())
    return;
  const IDENTINFO* idtinfo = IdentInfo();
  if (! infos[j].SetInfo(*pItem, idtinfo->index, idtinfo->styp, idtinfo->geom, token->pnt))
   return;
}

// Calculate intersections.
G2PointArray pnts(16);
AwCurveOperator operator(*(pModel->GetGeomParam()));
if (! operator.CalcIntersection2(infos, &pnts))
  return;
```

12.1.6 最短距離点

二つの図形アイテムの最短距離を計算します。

```
int MeasureDistance(int iswt, const AwPickInfo infos[], const G2Point& pner,
                    double* distance, G2Point points[]) const;
```

iswt	選択肢。
0	無限長。線分なら直線、円弧なら円として処理します。
1	有限長。
infos	二つの図形アイテムのピック情報。アイテムの幾何図形部分（POINT、LINE, CIRCLE または BEZIER のいずれか）を指していなければなりません。
pner	平行線または同心円の場合に最短位置を算出するのに使います。
distance	最短距離。
points	最短位置 (二つの図形アイテム上の点)。
戻り値	点数が配列の上限を超えたときは false を返します。
0	最短距離を計算した。
3	2 つの直線が交差してる（出力引数に交差角度を設定)。
他	計算できない。

例) 点 (20, 10) と曲線アイテムの最短距離。

```
AwPickInfo infos[2];
// Set point.
G2Point point(20.0, 10.0);
infos[0].SetPoint(point, point);

// Pick curve item.
const AwItem* pItem = PickItem(0, *token, 1);
if (pItem == NULL || ! pItem->IsCurve())
  return;
const IDENTINFO* idtinfo = IdentInfo();
```

```
  if (! infos[1].SetInfo(*pItem, idtinfo->index, idtinfo->styp, idtinfo->geom, token->pnt))
    return;

  // Calculate distance.
  G2Point pnts[2];
  double dist;
  AwCureveOperator operator(*(pModel->GetGeomParam()));
  if (! operator.MeasureDistancde(0, infos, point, &dist, pnts))
    return;
```

12.1.7 共通接線

二つの図形アイテムの共通接線を計算します。曲線アイテムのピック位置の幾何図形に対して接線を計算します。

```
bool CalcTangentLines2(const AwPickInfo infos[], G2LinePtrArray* pLineArray) const;
```

infos	二つの図形アイテムのピック情報。アイテムの幾何図形部分（POINT, CIRCLE または BEZIER のいずれか）を指していなければなりません。
pLineArray	線分を追加する G2LinePtrArray オブジェクトへのポインタ。
戻り値	線分数が配列の上限を超えたときは false を返します。

例) 点 (20, 10) から曲線アイテムへの接線。

```
  AwPickInfo infos[2];
  // Set point.
  G2Point point(20.0, 10.0);
  infos[0].SetPoint(point, point);
  // Pick curve item.
  const AwItem* pItem = PickItem(0, *token, 1);
  if (pItem == NULL || ! pItem->IsCurve())
    return;
  const IDENTINFO* idtinfo = IdentInfo();
  if (! infos[1].SetInfo(*pItem, idtinfo->index, idtinfo->styp, idtinfo->geom, token->pnt))
    return;

  // Calculate tangent.
  G2LinePtrArray lines(16);
  AwCurveOperator operator(*(pModel->GetGeomParam()));
  if (! operator.CalcTangentLines2(infos, &lines))
    return;
```

12.1.8 共通接円弧

二つの図形アイテムの共通接円弧を計算します。曲線アイテムのピック位置の幾何図形に対して接円弧を計算します。

```
bool CalcFillets2(const AwPickInfo infos[], double radius,
                  int indices[][2], G2CirclePtrArray* pArcArray) const;
```

infos	二つの図形アイテムのピック情報。アイテムの幾何図形部分（POINT, LINE、CIRCLE または BEZIER のいずれか）を指していなければなりません。
radius	接円の半径 (0 < radius)。
indices	円弧の接するサブレコードのインデックスの組。
indices[][0]	接円弧の始点が接する第一番目のアイテムののサブレコードインデックス。
indices[][1]	接円弧の終点が接する第二番目のアイテムののサブレコードインデックス。
pArcArray	円弧を追加する G2CirclePtrArray オブジェクトへのポインタ。
戻り値	円弧数が配列の上限を超えたときは false を返します。

例) 曲線アイテム間に半径 15 の接円弧。

```
// Pick two curve items.
AwPickInfo infos[2];
for (int j = 0; j < 2; ++j) {
    // ここで token を適切に設定するとする。
  const AwItem* pItem = PickItem(0, token, 1);
  if (pItem == NULL || ! pItem->IsCurve())
    return;
  if (j == 1 && pItem->GetItemId() == infos[0].GetItemId())
    return;
  const IDENTINFO* idtinfo = IdentInfo();
  if (! infos[j].SetInfo(*pItem, idtinfo->index, idtinfo->styp, idtinfo->geom, token->pnt))
   return;
}
// Calculate Fillet arcs.
G2CirclePtrArray arcs(16);
int indices[16][2];
AwCurveOperator operator(*(pModel->GetGeomParam()));
if (! operator.CalcFillets2(infos, 15.0, indices, &arcs))
  return;
```

12.2 AwPickInfo クラス

このクラスはアイテムのピック情報を保持します。
またピック情報ではない点やベクトルを保持するのにも使うことができます。この場合はアイテムに関する情報は持ちません。

12.2.1 コンストラクタ

```
AwPickInfo();
```

空のオブジェクトです。

12.2.2 Setter

アイテムのピック情報を設定します。

```
bool SetInfo(const AwItem& item, int index, const G2Geom& geom, const G2Point& location);
```

item	アイテムの参照。
index	ピックしたサブレコード・インデックス (0-)。
geom	ピックしたサブレコードから得た幾何図形オブジェクトの参照。
location	ピック位置。
戻り値	成功したとき true を返します。

```
bool SetInfo(const AwItem& item, int index, int type, const DGEOM& geom, const G2Point& location);
```

item	アイテムの参照。
index	ピックしたサブレコード・インデックス (0-)。
type	ピックしたサブレコードから得た幾何図形の種類（POINT、LINE、CIRCLE または BEZIER)。
geom	ピックしたサブレコードから得た幾何図形。
location	ピック位置。
戻り値	成功したとき true を返します。

幾何図形を設定します。

```
bool SetInfo(const G2Geom& geom, const G2Point& location);
```

geom	幾何図形オブジェクトの参照。
location	ピック位置。

点を設定します。
```
bool SetInfo(const G2Point& point, const G2Point& location);
```
point　　点。
location　　ピック位置。
戻り値　　成功したとき true を返します。

ベクトルを設定します。
```
bool SetInfo(const G2Vector& vector);
```
戻り値　　成功したとき true を返します。

12.2.3 Getter

このオブジェクトが持つ幾何図形の種類を得ます。
```
int GetGeomType() const;
```
戻り値　　ベクトルであれば G2Geom::VECTOR を返します。何も設定されていなければ 0 を返します。
```
bool HasCurve() const;
```
戻り値　　このオブジェクトが曲線を持つとき true を返します。
```
bool HasPoint() const;
```
戻り値　　このオブジェクトが点を持つとき true を返します。
```
bool HasVector() const;
```
戻り値　　このオブジェクトがベクトルを持つとき true を返します。

アイテム識別子を得ます。
```
int GetItemId() const;
```
戻り値　　アイテム識別子を返します。アイテムが設定されていなければ 0 を返します。

サブレコード・インデックスを得ます。
```
int GetSrIndex() const;
```
戻り値　　サブレコード・インデックスを返します。アイテムが設定されていなければ 0 を返します。

幾何図形を得ます。
```
const G2Geom* GetGeometry() const;
```
戻り値　　HasVector() が true なら null ポインタを返します。

ベクトルを得ます。
```
const G2Vector* GetVector() const;
```
戻り値　　HasVector() が false なら null ポインタを返します。

12.3 AwCurveItemIterator クラス

このクラスは曲線アイテムのイテレータです。このイテレータはアイテムの曲線サブレコードを見つけるとそれに対応する幾何図形を返します。
サブレコード選択マスクを使って、曲線サブレコードの種類、線種を絞ることができます。デフォルトは全ての曲線サブレコード・タイプ、表示可能なサブレコード線種が対象です。

```
const AwItem* pItem = pModel->GetPermItemDb(idptr);
if (pItem == NULL || ! pItem->IsCurve())
  return;
// イテレータを作る
AwCurveItemIterator iterator(*pItem);
// イテレートする
while (iterator.HasNext()) {
  const G2Curve* pCurve = iterator.Next();
```

```
    // do something.
  }
```

12.3.1 コンストラクタ

```
AwCurveItemIterator(const AwItem& item, bool bTrim = false,
                    int first = 0, int last = AwItem::MAX_INDEX);
```

item	曲線アイテムの参照。
bTrim	スプラインアイテムでのみ有効。トリムした両端の非表示部分が不要なとき true を設定する。
first	開始サブレコードのインデックス (0 <= first)。省略時はアイテムの先頭から開始します。
last	終了サブレコードのインデックス (first <= last)。省略時はアイテムの最後までイテレートします。

12.3.2 メソッド

次の幾何図形を得るためイテレータを進めます。

```
bool HasNext();
```

戻り値　次の幾何図形があるとき true を返します。false のときはイテレーション終了。

幾何図形を得る。

```
G2Curve* Next();
```

戻り値　G2Curve オブジェクトへのポインタを返します。HasNext() が true であれば必ず有効なポインタが得られます。そうでないときは null ポインタを返します。このオブジェクトは次の HasNext() メソッドが削除します。このポインタを使ってオブジェクトを削除 (delete) してはいけません。

このイテレータが処理している現在のサブレコードの情報を得ます。

```
int GetSrIndex() const;
```

戻り値　サブレコードのインデックスを返します。

```
int GetSrLineFont() const;
```

戻り値　サブレコードの線種を返します。

```
int GetSrLineWeight() const;
```

戻り値　サブレコードの線幅を返します。

12.4 AwMassProperty クラス

このクラスは幾何学諸元を計算します。

12.4.1 コンストラクタ

```
AwMassProperty(const AwGeomParam* pGeomParam);
```

pGeomParam　AwGeomParam オブジェクトへのポインタ。

12.4.2 アクセッサ

計算の基準となる座標系を設定します。省略時はモデル座標です。

```
void SetMatrix(const G2Matrix* pMatrix);
```

pMatrix　G2Matrix オブジェクトへのポインタ。

計算結果を得ます。

```
const double* GetResults() const;
```

戻り値　結果を持つ配列へのポインタ を返します。

12.4.3 面積

面積計算で与える領域は、交差の無い、閉じた形状でなければなりません。ひとつの閉じた曲線アイテム（円、スプライン、ストリング）を与えるか、ひとつの閉領域を構成する複数の曲線アイテム（線分、円弧など）を与えます。複数の曲線アイテムを与える場合、曲線アイテムは前後の曲線アイテムと接続していなければなりません。その中に逆向きの曲線アイテムがあってもかまいません。順方向の曲線アイテムのアイテム識別子は正の整数、逆向きの曲線アイテムは負の整数で識別します。

面積を計算します。

```
int CalcArea(AwItemListIterator* pListIterator);
```

pListIterator　閉領域を構成する曲線アイテムを持つイテレータ。
戻り値　計算できたら G2Math::CCw または G2Math::Cw を返します。
G2Math::CCw　領域外周の向きは反時計回り
G2Math::Cw　領域外周の向きは時計回り
その他　計算できなかった。

```
int CalcArea(const AwItem& item);
```

item　閉じた曲線アイテムの参照。
戻り値　計算できたら G2Math::CCw または G2Math::Cw を返します。
G2Math::CCw　領域外周の向きは反時計回り
G2Math::Cw　領域外周の向きは時計回り
その他　計算できなかった。

GetResults() メソッドで得る結果は次の通りです。

[0] 周長 (L)
[1] 面積 (A)
[2] X 軸に関する断面一次モーメント (Gx)
[3] Y 軸に関する断面一次モーメント (Gy)
[4] X 軸に関する断面二次モーメント (Ix)
[5] Y 軸に関する断面二次モーメント (Iy)
[6] 断面相乗モーメント (Ixy)
[7] 縁距離 (X 軸の最小座標)
[8] 縁距離 (Y 軸の最小座標)
[9] 縁距離 (X 軸の最大座標)
[10] 縁距離 (Y 軸の最大座標)

上記の値を使って下記のような諸元を算出できます。

重心　cx = Gy/A, cy = Gx/A
回転半径　rx = sqrt(Ix/A), ry = sqrt(Iy/A)
主軸　atan(-2Ixy, Ix-Iy) / 2

例

```
void uarea(const TOKEN& token) {
  // 境界アイテムをピック
  const AwItem* pItem = PickItem(1, token, 1);
  if (pItem == NULL)
    return;
  // アイテムタイプを確認
  if (pItem->GetType() < AwItem::CIRCLE || AwItem::STRING < pItem->GetType())
    return;
  // 閉じていることを確認
  G2Point pse[2];
  AwCureveOperator operator(*(pModel->GetGeomParam()));
```

```
    if (operator.GetEndPoints(*pItem, pse) != 2)
      return;
    if (pse[0].CalcLinfDistance(pse[1]) > 1.0e-3)
      return;
    // 計算
    AwMassProperty calculator(pModel->GetGeomParam());
    if (calculator.CalcArea(*pItem) == 0)
      return;
    // 結果
    const double* prop = calculator.GetResults();
    printf("Perimeter   L = %g", prop[0]);
    printf("Area        A = %g", prop[1]);
    printf("Moment     Gx = %g", prop[2]);
    printf("           Gy = %g", prop[3]);
    printf("Inertia    Ix = %g", prop[4]);
    printf("           Iy = %g", prop[5]);
    printf("          Ixy = %g", prop[6]);
    printf("Center     Cx = %g", prop[3] / prop[1]);
    printf("           Cy = %g", prop[2] / prop[1]);
    printf("Principal axis= %g",
           G2Math::ToDegree * 0.5 * atan2(-2.0 * prop[6], prop[4] - prop[5]));
}
```

12.4.4 回転体の体積

体積計算は、断面形状 (2D) を X- 軸 回りに回転してできる回転体の体積を計算します。断面形状は X- 軸と交差してはなりません。断面形状は、交差の無い形状でなければなりません。ひとつの曲線アイテム（線分、円弧、スプライン、ストリング）を与えるか、形状を構成する複数の曲線アイテム（線分、円弧など）を与えます。複数の曲線アイテムを与える場合、曲線アイテムは前後の曲線アイテムと接続していなければなりません。形状の中に逆向きの曲線アイテムがあってもかまいません。順方向の曲線アイテムのアイテム識別子は正の整数、逆向きの曲線アイテムは負の整数で識別します。

回転体の体積を計算します。

```
int CalcVolume(AwItemListIterator* pListIterator);
```

pListIterator　回転体の断面を構成する曲線アイテムを持つイテレータ。
戻り値　　計算できたら 0 を返します。

GetResults() メソッドで得る結果は次の通りです。

[0] 表面積
[1] 体積

12.5 AwCurveOffsetter クラス

このクラスは曲線アイテムをオフセットします。

12.5.1 コンストラクタ

```
AwCurveOffsetter(AwModel* pModel);
```

AwModel オブジェクトへのポインタを渡します。

12.5.2 条件

補間方法を取得します。

```
int  GetExtrapolation() const;
```

戻り値　　補間方法を返します。

0　　交点まで延長。
1　　交点まで延長 (cut 参照)。
2　　円弧補間（デフォルト）。

カットオフ値を取得します。補間方法（type==1）
```
double GetCutoffRatio() const;
```
戻り値　カットオフ値を返します。

補間方法を設定します。
```
void SetExtrapolation(int type, double cut = 0.0);
```
type　補間方法。
0　交点まで延長。
1　交点まで延長 (cut 参照)。
2　円弧補間（デフォルト）。
cut　交点まで延長 (type==1) の場合にできる鋭い突出を避けるため制限を設定します。cut はオフセット距離に対する比率です (1.0 <= cut <= 10.0)。突出は（cut * オフセット距離）以内になるよう先端を切断されます。

元の曲線の部分線種を保持するかどうかを取得／設定します。
```
bool IsKeepSrLinefont() const;
void SetKeepSrLinefont(bool bEnable);
```
保持するなら true。デフォルトは false。

オフセット曲線の自己交差部分を除去するかどうかを取得／設定します。
```
bool IsGougeRemoval() const;
void SetGougeRemoval(bool bEnable);
```
除去するなら true。デフォルトは true。

12.5.3 オフセット

曲線アイテムのオフセットを作成します。もとの曲線が閉じたストリングアイテムのときは、オフセットしてできたストリングアイテムも自動的に閉じます。オフセットはテンポラリアイテムとして作成されます。このメソッドはテンポラリアイテムの表示は行いません。

```
int CreateOffset(int ofsdir, double ofsdst, const AwItem& item, int index = -1);
```
ofsdir　オフセットを作る側。
G2Math::Left曲線アイテムの左側。
G2Math::Right　曲線アイテムの右側。
ofsdst　オフセット距離 (0 <= ofsdst)。
item　曲線アイテムの参照。
index　曲線アイテムの場合はこの引数は省略します (index == -1)。複合アイテムなどに含まれている曲線部分をオフセットする場合に指定します。曲線部分のサブレコードのどれかを指すインデックスを与えます (0 <= index)。このインデックスを含む曲線部分を見つけ出しオフセットを作成します。
戻り値　結果を示す整数を返します。
0　成功。
1　成功。アイテムタイプを AwItem::STRING に変更した。スプラインアイテムをオフセットした結果、補間のために直線サブレコードまたは円弧サブレコードが挿入されたときです。
2　計算できなかった。

例
```
// アイテムタイプを調べる
if (! pItem->IsCurve())
  return;
// オフセット
AwCurveOffsetter ofsetter(pModel);
```

```
if (ofsetter.CreateOffset(G2Math::Left, 5.0, *pItem))
  return;
```

12.6 AwSplineCreator クラス

このクラスはスプラインを計算します。
スプラインアイテムを作成する手順は次の通りです。

- 空のテンポラリアイテムを作る (スプラインアイテム)
- テンポラリアイテムにスプラインデータを追加する
- テンポラリアイテムを閉じる

12.6.1 コンストラクタ

AwSplineCreator();
コンストラクタ。

12.6.2 スプライン

指定した点列を通るスプラインをテンポラリアイテムに追加します。
スプラインの点数は 3 以上です。スプラインコマンドの最大点数は 512 ですので、それを超えないことを薦めます。

bool AddSpline(AwTempItem* pTempItem, const G2PointArray& points, bool cyclic);

pTempItem	スプラインを追加するテンポラリアイテムへのポインタ。
points	スプラインの通過点列を含む G2PointArray オブジェクトの参照。
cyclic	滑らかに閉じたスプラインにするとき true とします。その場合は始点と終点は離れていなければなりません。cyclic を false としたときは始点と終点は同一座標でもかまいません。その場合、閉じたスプラインですが、始点／終点では連続性がありません。
戻り値	テンポラリアイテムのサブレコードに追加したとき true を返します。

12.6.3 円錐曲線を近似

円錐曲線を近似するスプラインをテンポラリアイテムに追加します。
bool AddConic(AwTempItem* pTempItem, const G2Conic& conic);

pTempItem	スプラインを追加するテンポラリアイテムへのポインタ。
conic	円錐曲線オブジェクトの参照。
戻り値	テンポラリアイテムにサブレコードを追加したとき true を返します。

例）楕円を近似するスプラインアイテム。

```
// 楕円 長半径 50, 短半径 25, 中心点 (100,100), 長軸の向きは正 Y 軸方向
const G2Conic conic;
if (! conic.SetEllipse(50.0, 25.0, G2Point(100.0, 100.0), G2Vector::Yunit))
  return;
// テンポラリなスプラインアイテムを作る
AwTempItemDb* pTempItemDb = pModel->GetTempItemDb();
pTempItemDb->Init();
AwTempItem* pTempItem = pTempItemDb->OpenItem();
if (pTempItem == NULL)
  return;
pTempItem->SetType(AwItem::SPLINE);
AwSplineCreator creator;
if (creator.AddConic(pTempItem, conic))
```

```
    pTempItem->Close();
  else
    pTempItemDb->Init();
```

12.7 関数

● 関数一覧

関数名	機能
gmsptcrv	曲線アイテムの２分割

12.7.1 曲線アイテムを指定位置で２分割

曲線アイテムを指定位置で２分割します。

【呼出し形式】

```
int gmsptcrv(const Gm_brkpos* brkpos, int idptr)
```

【入力引数】

brkpos　分割位置を指示するデータ

brkpos.idptr	アイテムの識別子
brkpos.snum	サブレコードの相対番号
brkpos.type	図形要素の種類。SITPOINT, SITLINE, SITARC, SITBZCRV
brkpos.pnt	図形要素をピックした位置 (DPOINT)
brkpos.geom	図形要素データ (DGEOM)

これらのデータはアイテムを分割する位置を与えるもので、Ident00 の副作用として得られる。

idptrs　分割されるアイテムの識別子

【戻り値】

0　：正常終了
1　：エラー

注意

(1) 新しくできたアイテムはデータベースに格納される。
もとのアイテムは消去する。

(2) 分割位置指定が点のときは投影点で２分割する。
分割位置指定が曲線のときは交点で２分割する。

例 . 分割位置を指示するデータの設定方法

```
DPOINT p = {50.0, 50.0};
int idptr = 1;

Gm_brkpos brkpos;

if (boundary is point) {
  brkpos.geom.pnt = p;
  brkpos.idptr = 0;
```

```
    brkpos.snum = 0;
    brkpos.type = AwSr::POINT;
    brkpos.pnt = brkpos.geom.pnt;
  } else if (boundary is picked curve) {
    TOKEN token;
    tknclear(&token);
    token.typ = TknDIG;
    token.pnt = p;
    if (IdentItem(1, &token, 1) <= 0)
      return;
    const IDENTINFO* info = IdentInfo();
    brkpos.idptr = info[0].idptr;
    brkpos.snum = info[0].index;
    brkpos.type = info[0].styp;
    brkpos.geom = info[0].geom;
    brkpos.pnt = p;
  } else {
    return;
  }
  gmsptcrv(&brkpos, idptr);
```

第 13 章　製図アイテム

この章は製図アイテムを扱うクラスを説明します。製図アイテムとは以下のアイテムをいいます。

AwItem::TEXT	グラフィックテキストアイテム
AwItem::MARK	マークアイテム
AwItem::DIMENSION	寸法アイテム
AwItem::GTOL	幾何公差アイテム
AwItem::XHT	ハッチングアイテム
AwItem::AFL	塗りつぶしアイテム

13.1 AwNoteCreator クラス

このクラスはグラフィックステキストアイテムを作成します。グラフィックテキストにはジェネラルノート／ラベル、リファランスラベル／ラベルがあります。いずれもアイテムタイプは AwItem::TEXT です。
このクラスのメソッドはコンストラクタで与えられた製図定数を使います。製図定数を変更することはありません。一時的に変更しても、必ず元の値に復元します。このクラスのメソッドはグラフィックテキストをテンポラリアイテムに追加します。

13.1.1 コンストラクタ

```
AwNoteCreator(AwDrafParam* pDrafParam);
```
AwDrafParam オブジェクトへのポインタを渡します。

13.1.2 メソッド

ジェネラルノートをテンポラリアイテムに追加します。
```
void AddGeneralNote(AwTempItem* pTempItem, const std::string& text,
                    const G2Point& porg);
```
pTempItem	ジェネラルノート・インスタンスを追加するテンポラリアイテムへのポインタ。
text	文字列 (空文字列でないこと)。
porg	文字列の記入位置。

ジェネラルノートをテンポラリアイテムに追加します。文字列を指定した矩形に収める機能があります。
```
void AddGeneralNote(AwTempItem* pTempItem, const std::string& text,
                    int count, const G2Point points[],
                    int mode = 0, bool bFreeAngle = false);
```
pTempItem	ジェネラルノート・インスタンスを追加するテンポラリアイテムへのポインタ。
text	文字列 (空文字列でないこと)。

count	入力点数 (2 または 3)。
points	点列。points[0] は文字列の記入位置。文字列角度は points[0] と points[1] を結ぶ線分に合わせます。points[2] は文字高さを決めるのに使います（mode==2,3 のとき）。
mode	文字高さの決め方を指定します。
0	AwDrafParam の GetCharHeight() を使う。
1	2 つの直線が交差してる（出力引数に交差角度を設定）。
2	文字列の高さが 3 点で作る矩形に収まる文字高さを計算する。
3	文字列の横幅、高さの両方が 3 点で作る矩形に収まる文字高さを計算する。
bFreeAngle	文字列角度を通常の文字列角度の範囲に収めるかどうか指定します。
false	通常の文字列角度 (-85 < angle <= 95 度)。
true	任意の角度 (-180 < angle <= 180 度)。

ジェネラルラベルをテンポラリアイテムに追加します。

```
bool AddGeneralLabel(AwTempItem* pTempItem, const std::string& text,
                     int count, const G2Point points[]);
```

pTempItem	ジェネラルラベル・インスタンスを追加するテンポラリアイテムへのポインタ。
text	文字列 (空文字列でないこと)。
count	入力点数 (2 <= count <= 15)。
points	点列。最初の点が引き出し線の矢印が付く方です。
戻り値	成功したら true を返します。

リファランスラベルをテンポラリアイテムに追加します。

```
bool AddReferenceLabel(AwTempItem* pTempItem, const std::string& text,
                     int count, const G2Point points[]);
```

pTempItem	リファランスラベル・インスタンスを追加するテンポラリアイテムへのポインタ。
text	文字列 (空文字列でないこと)。
count	入力点数 (1 <= count <= 15)。
points	点列。一点だけのときは、引き出し線なしのリファランスノートになります。2 点以上のときは最初の点が引き出し線の矢印が付く方です。
戻り値	成功したら true を返します。

リファランスノートをテンポラリアイテムに追加します。

```
void AddReferenceNote(AwTempItem* pTempItem, const AwSrMark& srMark,
                     const AwSrText& srText);
```

pTempItem	リファランスノート・インスタンスを追加するテンポラリアイテムへのポインタ。
srMark	風船を表すマーク・サブレコード。
srText	文字列 (空文字列でないこと) を表すテキスト・サブレコード。

13.2 AwDimCreator クラス

このクラスは寸法アイテムを作成します。
長さ寸法は寸法値倍率を持ちます。これは、鋳物などの収縮を見込んで若干大きめな寸法値を記入したいが、製品形状を実際に拡大するのではなく寸法値だけを変えて記入したいときに使用します。
寸法作成メソッドに寸法アイテムの文字列（寸法値など）を渡すには後述する AwDimText クラスを使います。

13.2.1 コンストラクタ

```
AwDimCreator(AwTempItemDb* pTempItemDb);
```

AwTempItemDb オブジェクトへのポインタを渡します。

13.2.2 Setter

長さ寸法の寸法値測定基準ベクトルを設定します。水平／垂直／平行寸法いずれの場合でも水平方向を向くベクトルを設定します。設定しなければモデル座標 X 軸 (1,0) です。

```
void SetHorizon(const G2Vector& xaxis);
```

xaxis　　水平寸法の向きを与えるベクトル。

13.2.3 単独の寸法

寸法アイテムを単独で作成する手順は次の通りです。

- **寸法文字列を設定する (AwDimText)**
- **空のテンポラリアイテムを作る (寸法アイテム)**
- **テンポラリアイテムに寸法データを追加する**
- **テンポラリアイテムを閉じる**

アイテムに角度寸法を追加します。

```
bool AddDimAngular(AwTempItem* pTempItem, int subtype, short iswt,
                   const G2Point& vertex, const G2Line& lineStart, const G2Line& lineEnd,
                   const G2Point& porg, const AwDimText& dimText);
```

pTempItem	寸法を追加するテンポラリアイテムへのポインタ。
subtype	寸法サブタイプを示す整数。通常は AwSrDimension::DS_SINGLE。
iswt	オプション指定。指定しないときは 0 とします。
bit 1	第一寸法補助線を抑止 (1：抑止)。
bit 2	第二寸法補助線を抑止 (1：抑止)。
bit 3	第一外部寸法線を抑止 (1：抑止)。
bit 4	第一内部寸法線を抑止 (1：抑止)。
bit 5	第二内部寸法線を抑止 (1：抑止)。
bit 6	第二外部寸法線を抑止 (1：抑止)。
bit 9-11	寸法値の位置制御（0：指定位置、1：中央、2：納まるなら中央)。
vertex	角度寸法の基準点。
lineStart	寸法測定開始線。
lineEnd	寸法測定終了線。
porg	寸法値の位置。
dimText	寸法値。
戻り値	テンポラリアイテムにサブレコードを追加したら true を返します。

寸法測定線分には下記の制約があり、それを満たさないときの結果は不定です。

- **線分は有効な長さを持つこと**
- **線分は基準点を通過する直線上にあること**
- **線分は基準点をまたがってはならない**
- **角度は基準点からみて、開始線から終了線へ反時計回りに測定する**

アイテムに長さ寸法を追加します。

```
bool AddDimLinear(AwTempItem* pTempItem, int form, int subtype, short iswt,
                  double scale, const G2Point points[], const AwDimText& dimText);
```

pTempItem	寸法を追加するテンポラリアイテムへのポインタ。
form	寸法値測定タイプを示す整数（1：水平、2：垂直、3：平行)。
subtype	寸法サブタイプを示す整数。通常は AwSrDimension::DS_SINGLE。
iswt	オプション指定。指定しないときは 0 とします。
bit 1	第一寸法補助線を抑止 (1：抑止)。
bit 2	第二寸法補助線を抑止 (1：抑止)。
bit 3	第一外部寸法線を抑止 (1：抑止)。
bit 4	第一内部寸法線を抑止 (1：抑止)。
bit 5	第二内部寸法線を抑止 (1：抑止)。

bit 6　第二外部寸法線を抑止 (1：抑止)。
bit 9-11　寸法値の位置制御（0：指定位置、1：中央、2：納まるなら中央)。
scale　寸法値倍率 (0.0 < scale)。
points　二つの寸法記入点と寸法値の位置 (points[3])。
dimText　寸法値。
戻り値　テンポラリアイテムにサブレコードを追加したら true を返します。

アイテムに直径寸法を追加します。

```
bool AddDimDiameter(AwTempItem* pTempItem, short iswt, double scale, const G2Point& center,
                    const G2Point points[], int count, const AwDimText& dimText);
```

pTempItem　寸法を追加するテンポラリアイテムへのポインタ。
iswt　オプション指定。寸法補助線抑止を必ず設定します。
bit 1　第一寸法補助線を抑止 (1：抑止)。
bit 2　第二寸法補助線を抑止 (1：抑止)。
bit 3　第一外部寸法線を抑止 (1：抑止)。
bit 4　第一内部寸法線を抑止 (1：抑止)。
bit 5　第二内部寸法線を抑止 (1：抑止)。
bit 6　第二外部寸法線を抑止 (1：抑止)。
scale　寸法値倍率 (0.0 < scale)。
center　円中心点。
points　引き出し線の点列。最初の点は円周になければなりません。
count　点数は 2-3。ただし ANSI の時は 2 点のみ。
dimText　寸法値。
戻り値　テンポラリアイテムにサブレコードを追加したら true を返します。

アイテムに半径寸法を追加します。

```
bool AddDimRadius(AwTempItem* pTempItem, double scale, const G2Point& center,
                  const G2PointArray& points, int type, const G2Point& pend,
                  const AwDimText& dimText);
```

pTempItem　寸法を追加するテンポラリアイテムへのポインタ。
scale　寸法値倍率 (0.0 < scale)。
center　円中心点。
points　主引き出し線の点列。最初の点は円周になければなりません。点数は 2-16。
type　引き出し線オプション。指定しないときは 0 とします。
1　主引き出し線が円外なら、円内側に矢印なしの副引き出し線をつける。
4　主引き出し線が円内で点数２なら、主引き出し線終点を円中心にする。
pend　円内側引き出し線の終端点（type = 1 の時だけ参照)。
dimText　寸法値。
戻り値　テンポラリアイテムにサブレコードを追加したら true を返します。

例〕水平長さ寸法アイテムを作る。

```
const G2Point points[] = {
  G2Point(-50.0,  0.0),
  G2Point( 50.0,  0.0),
  G2Point(  0.0, 50.0)
};
const double scale = 1.0; // Dimension value scale.

// Set dimension value.
double value = points[0].CalcDistance(points[1]);
AwDimText dimText(pModel->GetDrafParam(), 1);
dimText.SetValue(AwDimText::LINEAR, value * scale);

// Create new temporary item.
AwTempItemDb* pTempItemDb = pModel->GetTempItemDb();
AwTempItem* pTempItem = pTempItemDb->OpenItem();
```

```
if (pTempItem == NULL)
  return;
pTempItem->SetType(AwItem::DIMENSION);

// Add dimension data.
AwDimCreator creator(pTempItemDb);
if (creator.AddDimLinear(pTempItem, 1, AwSrDimension::DS_SINGLE, 0, scale, points, dimText)
    && pTempItem->Close()) {
  // OK
  pTempItemDb->StoreItems();
} else {
  // FAIL
  pTempItemDb->RemoveLastItem();
}
```

13.2.4 複数の寸法

累進寸法、オーディネイト寸法を作成するメソッドは複数のテンポラリアイテム（寸法アイテム）を作成します。累進寸法の起点を示す寸法は寸法参照点間の距離がゼロになる特異な寸法です。このような場合以外は「ゼロ寸法」は想定していません。

累進長さ寸法アイテムを作成します。

```
int CreateRunningLinear(int form, bool ispdim, int numdim,
                        const double scales[], const G2Point points[],
                        const G2Point& porg, const AwDimText texts[]);
```

form	寸法値測定タイプを示す整数（1：水平、2：垂直）。
ispdim	起点の寸法の抑止。
false	起点寸法も作る。
true	起点寸法は作らない。
numdim	作成する寸法アイテム数（1 - 127）。
scales	寸法値倍率 (0.0 < scale) の配列。(scales[numdim+1])
points	寸法記入点列。(points[numdim+1])。points[0] は起点、points[1] - [numdim] は寸法参照点です。
texts	寸法値の配列。(texts[numdim+1])
戻り値	新たに作成したテンポラリアイテム数を返します。起点寸法を作ると numdim+1、起点寸法を作らないと numdim。

引数 points、scales、texts は numdim+1 の長さの配列です。points[i]、scales[i]、texts[i] が対応します。最初の要素が寸法の起点、残りが寸法参照点です。寸法参照点は寸法値測定基準ベクトルに沿って順番に並んでいるものとします。

n1 > n2 > . . . > nm < p1 < p2 < . . . < pm.

n1 - nm は参照点に対して基準ベクトルと逆方向にあり、参照点に遠い点から近く点の順、p1 - pm は参照点に対して基準ベクトルと同方向にあり、参照点に近い点から遠く点の順に並べます。

オーディネイト寸法アイテムを作成します。

```
int CreateOrdinate(int form, int numdim,
                   const double scales[], const G2Point points[],
                   const G2Point& porg, const AwDimText texts[]);
```

form	寸法値測定タイプを示す整数（1：横座標、2：縦座標）。
numdim	作成する寸法アイテム数（1 - 127）。
scales	寸法値倍率 (0.0 < scale) の配列。(scales[numdim])
points	寸法記入点列。(points[numdim+1])。points[0] は起点、points[1] - [numdim] は寸法参照点です。
texts	寸法値の配列。(texts[numdim])
戻り値	新たに作成したテンポラリアイテム数を返します。numdim。

引数 points は numdim+1 の長さの配列です。引数 scales、texts は numdim の長さの配列です。
points[i+1]、scales[i]、texts[i] が対応します。
最初の要素が寸法の起点、残りが寸法参照点です。寸法参照点は寸法値測定基準ベクトルに沿って順番に並んでいるものとします。

n1 > n2 > . . . > nm < p1 < p2 < . . . < pm.

n1 - nm は参照点に対して基準ベクトルと逆方向にあり、参照点に遠い点から近く点の順、p1 - pm は参照点に対して基準ベクトルと同方向にあり、参照点に近い点から遠く点の順に並べます。

13.3 AwDimText クラス

このクラス は寸法の値、許容差などの文字列を処理するクラスです。また寸法アイテム作成メソッドの引数としても使用します。

オブジェクトは寸法値文字列、許容差文字列、付加文字列を持ちます。

寸法値文字列は空文字列は許さず、クリアすると一文字の空白文字を設定します。寸法値の文字列は前文字列、寸法数値、後文字列の三つの部分から成るひとつの文字列です。前文字列は ”2 X” など寸法値の前に、後文字列は嵌め合い公差クラス ”F7” など寸法数値の後に記入する文字列です。前文字列と後文字列が空文字列、つまり寸法数値だけのことは普通です。寸法数値は前文字列や後文字列と区別するため特別な文字で囲みます。この処理はこのクラスのメソッド内で行います。

許容差文字列は ± 許容差、上許容差、下許容差の三つの文字列です。± 許容差を設定すると、上許容差と下許容差は無効となります。許容差文字列は空文字列でもかまいません。

付加文字列は許容差の後に注記を追加するためのものです。許容差を括弧で囲む時も使用します。寸法値の文字列の最後に '(' 文字、ポスト文字列の先頭を ')' 文字にします。付加文字列は空文字列でもかまいません。

13.3.1 定数

定数名	説明
MAX_TEXT_LENGTH	寸法値文字列、付加文字列の最長（バイト）
MAX_TOL_LENGTH	寸法許容差の文字列の最長（バイト）

定数名	説明
SIMPLE	単純変換（無名数）
LINEAR	長さ寸法値
ANGULAR	角度寸法値
RADIUS	半径寸法値
DIAMETER	直径寸法値
BILATERAL	長さ寸法の ± 許容差
BILATERAL	長さ寸法の上／下許容差（数値の前に＋または一符号）

定数名	説明
ADD_SQUAR	補助記号□付き 長さ寸法値
ADD_CONSTANT	” 一定 ” 付き 長さ寸法値
CHAMFER	45 度隅切り寸法値（寸法数値の前に C）
ANG_BILATERAL	角度寸法の ± 許容差
ANG_TOLERANCE	角度寸法の上／下許容差
ADD_DIAMETER	直径記号付き 長さ寸法値
COORDINATE	座標寸法値 ″(X,Y)″

13.3.2 コンストラクタ

このクラスのメソッドは寸法数値の小数桁数などのパラメータを持つ AwDrafParam オブジェクトを参照します。長さ単位は長さと座標を処理するとき使用します。

```
AwDimText(const AwDrafParam* pDrafParam, int unit);
```
AwDrafParam オブジェクトへのポインタ (pDrafParam)、モデルの長さ単位 (unit) を渡します。

13.3.3 Getter

このクラスは寸法値と寸法許容差の文字列を持ちます。
```
const std::string& GetValue() const;
```
戻り値　寸法値文字列を返します。
```
const std::string& GetPosttext() const;
```
戻り値　付加文字列を返します。
```
const std::string& GetBilateralTolerance() const;
```
戻り値　± 寸法許容差を返します。
```
const std::string& GetUpperTolerance() const;
```
戻り値　上寸法許容差を返します。
```
const std::string& GetLowerTolerance() const;
```
戻り値　下寸法許容差を返します。

13.3.4 Setter

この以下のメソッドは引数で渡された文字列をそのまま設定します。寸法値文字列の規則を適用した文字列を設定するのであれば後述のメソッドを使います。

寸法値文字列を設定します。
```
void SetValue(const std::string& text);
```
付加文字列を設定します。
```
void SetPosttext(const std::string& text);
```
± 寸法許容差の文字列を設定します。
```
void SetBilateralTolerance(const std::string& text);
```
上寸法許容差の文字列を設定します。
```
void SetUpperTolerance(const std::string& text);
```
下寸法許容差の文字列を設定します。
```
void SetLowerTolerance(const std::string& text);
```

13.3.5 寸法値の文字列

寸法値文字列に規則を適用した文字列を設定するメソッドです。

寸法の文字列生成における特殊文字

特殊文字	説明
*	寸法数値の位置を示す文字。寸法数値で置き換えられます。最初に現れた "*" だけが有効です。
&	許容差の位置を示す文字。寸法値文字列と付加文字列の区切りを意味します。最初に現れた "&" だけが有効です。
\c	文字 c。特殊文字 *, & をエスケープし通常文字とします。* は *、\& は &、\\ は \ です。
寸法数値	メタ文字 (\MD) と (\MZ) に囲まれた寸法値。

入力文字列の解釈規則。

- 寸法数値を含むときは、寸法数値は変更しないでそのまま使う。
- 文字 * を含み寸法数値がないときは、文字 * が寸法数値で置き換えられる。
- 文字 * も寸法数値もないときは、寸法値文字列は入力文字列で置き換られ寸法数値を含まない。

このオブジェクトに寸法値文字列と付加文字列を設定します。

```
bool SetValue(const std::string& text, int form, const double dimval[]);
```

text	寸法値文字列と付加文字列を連結した文字列。文字 *、& は特別な意味を持ちます。
form	寸法数値の書式を指定する整数。
dimval	寸法値。座標寸法の場合 (form == COORDINATE) は value[0] に X 座標、value[1] に Y 座標を設定します。それ以外は value[0] に寸法値を設定します（0 <= value[0]）。
戻り値	成功したら true を返します。

```
bool SetValue(const std::string& text, int form, double dimval);
```

text	寸法値文字列と付加文字列を連結した文字列。文字 *、& は特別な意味を持ちます。
form	寸法数値の書式を指定する整数。form != COORDINATE。
dimval	寸法値（0 <= dimval）。
戻り値	成功したら true を返します。

このオブジェクトに寸法値文字列を設定し、付加文字列をクリアします。
SetValue("*", form, &value) の短縮形です。

```
bool SetValue(int form, double value);
```

form	寸法値の書式を指定する整数。form != COORDINATE。
value	寸法値（0 <= value）。
戻り値	成功したら true を返します。

このオブジェクトに座標寸法の寸法値文字列を設定し、付加文字列をクリアします。
SetValue("*", AwDimText::COORDINATE, 座標) の短縮形です。

```
bool SetValue(const G2Point& point);
```

point	座標。
戻り値	成功したら true を返します。

このオブジェクトの寸法値文字列、許容差文字列、付加文字列をクリアします。

```
void Clear();
```

例)

```
AwDimText text(pModel->GetDrafParam(), 1);
text.SetValue("ABC*DEF&GHI", AwDimText::LINEAR, 100.0);
```

結果は寸法値文字列 "ABC\MD100\MZDEF", 付加文字列 "GHI" になります。

13.3.6 許容差の文字列

このオブジェクトに± 寸法許容差の文字列を設定します。
```
bool SetBilateralTolerance(int form, double tol);
```
form	許容差の書式を指定する整数。
tol	許容差。
戻り値	成功したら true を返します。

このオブジェクトに上寸法許容差の文字列を設定します。
```
bool SetUpperTolerance(int form, double tol);
```
form	許容差の書式を指定する整数。
tol	許容差。
戻り値	成功したら true を返します。

このオブジェクトに下寸法許容差の文字列を設定します。
```
bool SetLowerTolerance(int form, double tol);
```
form	許容差の書式を指定する整数。
tol	許容差。
戻り値	成功したら true を返します。

13.3.7 単純文字列

数値を書式に従って文字列に変換します。
```
static bool ToString(int form, double scalar, std::string* string,
                     const AwDrafParam& drafParam, int unit);
```
form	数値の書式を指定する整数。
scalar	数値。
string	数値を変換した文字列を追加する std::string オブジェクトへのポインタ。
drafParam	AwDrafParam オブジェクトの参照。
unit	モデルの長さ単位。
戻り値	成功したら true を返します。

座標を文字列に変換します。
```
static bool ToString(const G2Point& point, std::string* string,
                     const AwDrafParam& drafParam, int unit);
```
point	座標 (x,y)。
string	座標を変換した文字列を追加する std::string オブジェクトへのポインタ。
drafParam	AwDrafParam オブジェクトの参照。
unit	モデルの長さ単位。
戻り値	成功したら true を返します。

数値を文字列に変換します。
```
static bool ToString(double scalar, int maccu, int mtdlm, int mfdlm, int mtzer, std::string* string)
```
scalar	数値 (abs(scalar) ≦ 1.0e16)。
maccu	十進少数桁数 (0 ≦ maccu ≦ 12)。
mtdlm	十進整数部の 3 桁区切り記号。 0 = なし、1 = カンマ、2 = 空白、3 = ピリオド。
mfdlm	十進小数点記号。 0 = ピリオド、1 = カンマ
mtzer	十進表示の小数部の「後ろのゼロを除去」。 0 = 後ろのゼロを除去する、1= 後ろのゼロも表示する。
string	数値を変換した文字列を追加する std::string オブジェクトへのポインタ。
戻り値	成功したら true を返します。

13.4 関数

● 関数一覧

関数名	機能
gmgenafl	ぬりつぶしアイテムを作成する。
Drfulab	ラベルを作る。

13.4.1 ラベル作成

ラベルを作ります。

【呼出し形式】

```
void Drfulab(const char* pfxtxt, int num, int lnum, char* label)
```

【入力引数】

pfxtxt　接頭語 (Null-char terminated)
　　接頭語なしにするには空文字列か null ポインタを渡す。
num　番号。
　　num < 0　接頭語と番号の間にハイフン (-) を置く
　　num ≧ 0　接頭語と番号の間にハイフン (-) なし
lnum　番号の桁数
　　lnum == 0　番号なし
　　1 ≦ lnum ≦ 5　番号の前の零 (0) を除去しない。
　　-5 ≦ lnum ≦ -1　番号の前の零 (0) を除去する。

【出力引数】

abtxt　ラベル (接頭語＋番号)(Null-char terminated)

例
つぎのように４種類のラベルを作ることができる。
接頭語 "PNT" で、
番号が正 (num == 1) の場合
　桁数 lnum == 3　ラベル "PNT001"
　　lnum == -3　ラベル "PNT1"
番号が負 (num == -1) の場合
　桁数 lnum == 3　ラベル "PNT-001"
　　lnum == -3　ラベル "PNT-1"

13.4.2 塗り潰しアイテムを作成

塗り潰しアイテム (Area fill item) を作成します。

【呼出し形式】

```
int gmgenafl(const int idptrs[], int nitm, const float* aflprm, int dupswt)
```

【入力引数】

idptrs　塗り潰しアイテムの境界を構成するアイテムのアイテム識別子の並び
　　(int idptrs[nitm])。

境界を構成するアイテムはそれぞれ閉曲線でなければならない。円、閉自由曲線、閉ストリングアイテムまたは塗り潰しアイテムでもよい。

nitm　境界を構成するアイテムの数。

aflprm　塗り潰しパラメータ。

（これは V17 のサブレコード 18 のデータに相当するもの）

aflprm[0]　パターンのパラメータ。
- bit 1 - 2: パターン種類（0: デバイスのパターン、1: マーク、2: 文字）。
- bit 9 - 16: テキストフォント番号（パターン種類が文字の場合のみ）。
  - ASCII char 1-99, 102-109。
  - Japanese char 1-9。
- bit 17 - 32: パターン種類に依存するパターン番号。
  - 0 : デバイスのパターン番号（0 ー）
  - 1 : マーク番号（1 ー）
  - 2 : 文字コード（EUC で 2 バイト以内）。

aflprm[1]　パターンの配置角度（0-360 度）

aflprm[2],[3]　パターンの配置基準点。

aflprm[4],[5]　パターンの大きさ。負の値はパターンを反転する。

aflprm[6],[7]　パターンの間隔

dupswt　塗り潰しアイテム作成後、もとのアイテムを削除するかどうか。
- FALSE : 元のアイテムは削除する。
- TRUE : 元のアイテムを残す。

【戻り値】

0 : 正常

1 : エラー

第 14 章　構造化アイテム

この章は構造化アイテムについて説明します。構造化アイテムとは、図形アイテムや製図アイテムを内部に含む以下のアイテムをいいます。

AwItem::COMPOSITE　コンポジットアイテム
AwItem::APG　パラメトリックアイテム
AwItem::SYMBOL　シンボルアイテム
AwItem::SUBMODEL　サブモデルアイテム

もうひとつはアソシエイションアイテムです。

AwItem::ASSOCIATION アソシエイションアイテム

アソシエイションアイテムはアイテム間の依存関係を持つ特別なアイテムです。このアイテムは依存関係のあるアイテムへの参照（実際にはアイテム識別子）を持つことが本質で、表示する図形や文字列などは持ちません。

14.1 AwCompositeCreator クラス

このクラス はコンポジットアイテムを作成、変更するメソッドを持ちます。アイテムタイプはコンポジット (AwItem::COMPOSITE) です。コンポジットアイテムにはアソシエイションアイテム、イメージアイテムを含めることはできません。コンポジットアイテムを別のコンポジットアイテムの構成要素として含めることもできます。ただし、コンポジットアイテムの構成要素は階層を持つことはできず、すべて並列になります。

14.1.1 コンストラクタ

```
AwCompositeCreator(AwTempItemDb* pItemDb);
```

AwTempItemDb オブジェクトへのポインタを渡します。

14.1.2 要素を追加

テンポラリアイテムにコンポジットの要素を追加します。

```
bool AddComposite(AwTempItem* pItem, AwItemListIterator* pListIterator,
                  bool keepIds);
```

pTempItem	コンポジット要素を追加するテンポラリアイテムへのポインタ。
pListIterator	コンポジットに含めるアイテムを得る AwItemListIterator オブジェクトへのポインタ。
keepIds	コンポジットに含めたアイテムのアイテム識別子を保持するとき true を設定します。
戻り値	成功したら true を返します。要素を追加中に、アイテムのサイズの上限（サブレコード数、総データサイズ）を超えたら中断し、false を返します。このときテンポラリアイテムは不完全な状態です。

AddComposite() メソッドが成功したときに以下の情報を得る事ができます。
テンポラリアイテムに要素として追加したアイテムの数を得ます。
```
int GetItemCount() const;
```
戻り値　　処理したアイテム数を返します。

テンポラリアイテムに要素として追加したアイテムのアイテム識別子を得ます。AddComposite() メソッドの keepIds を true にした時だけ有効です。
```
const AwItemIdArray& GetSourceItems() const;
```
戻り値　　アイテム識別子を持つ AwItemIdArray オブジェクトの参照を返します。

コンポジットに追加した元のアイテムはそのまま残っています。元のアイテムを削除するのに GetSourceItems() メソッドで得たアイテム識別子が使えます。

14.2 シンボルインスタンス

14.2.1 テンポラリなシンボルアイテムの作成

テンポラリなシンボルアイテムを作成する。

【呼出し形式】

```
int SymPutl(const char* name, const DPOINT* org, const double scf[], double ang)
```

【入力引数】

name　　シンボル名 (Null-char terminated)。シンボル名に、シンボルファイルの拡張子 ".SYM" を含める必要はない。
org　　シンボル配置位置 (シンボル原点の位置)
scf　　シンボル縮尺値 (double scf[2])
　scf[0]　:　X 縮尺値 (シンボルの X 軸方向に対する縮尺値)。
　　scf[0] が負である場合には、シンボルの Y 軸に対して反転する。
　scf[1]　:　Y 縮尺値 (シンボルの Y 軸方向に対する縮尺値)
　　scf[1] が負である場合には、シンボルの X 軸に対して反転する。
　拡大／縮小しないときは、scf[0] = scf[1] == 1.0 とする。
ang　　シンボル回転角度 (単位 : 度)

【戻り値】

0　　: **正常終了**

注意
この関数は、指示された配置パラメターをもとに、テンポラリなシンボルアイテムを追加する。シンボルアイテムを作るにはつぎのようにする。
```
AwTempItemDb* pTempItemDb = pModel->GetTempItemDb();
pTempItemDb->StoreItems();
if (SymPutl(name, org, scf, ang) == 0)
  pTempItemDb->StoreItems();
```

14.2.2 シンボルアイテムのヘッダー情報を参照

シンボルアイテムのヘッダー情報を参照する。

【呼出し形式】

```
int SymInpHedl(int idptr, char* name, short idate[], DPOINT* org, double scf[],
               double* ang, DPOINT box[]);
```

【入力引数】

idptr　シンボルアイテムのアイテム識別子

【出力引数】

name　シンボル名 (Null-char terminated)。
idate　シンボルアイテム作成日時
idate[0] : 日、idate[1] : 月、idate[2] : 年
idate[3] : 時、idate[4] : 分、idate[5] : 秒
org　シンボル配置位置 (シンボル配置位置 : シンボル原点の位置)
scf　シンボル縮尺値 (double scf[2])
scf[0] : X 縮尺値 (シンボルの X 軸方向に対する縮尺値)。負の場合は、シンボルの Y 軸に対して反転していることを示す。
scf[1] : Y 縮尺値 (シンボルの Y 軸方向に対する縮尺値)。負の場合は、シンボルの X 軸に対して反転していることを示す。
ang　シンボル回転角度 (単位 : 度)
box　シンボルボックス
box[0] : シンボルボックスの左下の点
box[1] : シンボルボックスの右下の点
box[2] : シンボルボックスの左上の点
box[3] : シンボルボックスの右上の点

【戻り値】

0　: 正常終了

14.3 サブモデルインスタンス

14.3.1 テンポラリなサブモデルアイテムを作成

テンポラリなサブモデルアイテムを作成する。

【呼出し形式】

```
int Sub101(const char* name, int keymsk, int keydim, int keydrf, int ipicfrm,
           const DPOINT& org, const double scf[], double ang, int itmtyp)
```

【入力引数】

name　サブモデル名 (Null-char terminated)。サブモデル名に、モデルファイルの拡張子 ".MDL" を含める必要はない。
keymsk　アイテム選択マスク制御スイッチ
0 : 選択マスクを無視する。
1 : 一時的な選択マスクで指示されたアイテムだけをサブモデルアイテム化する。
keydim　寸法の調整制御スイッチ
0 : サブモデルに含まれる寸法を入力パラメータ (ORG, SCF, ANG) に合わせて調整しない。
1 : サブモデルに含まれる寸法を入力パラメータ (ORG, SCF, ANG) に合わせて調整する。
keydrf　製図図形要素の縮尺制御スイッチ
0 : 製図図形要素は縮尺しない。幾何図形要素だけをサブモデル縮尺値 (scf) に基づいて縮尺する。
1 : 製図図形要素も幾何図形要素と同様にサブモデル縮尺値 (scf) に基づいて縮尺する。

ipicfrm　サブモデル配置ピクチャ番号。サブモデル名で指定されたモデルファイルのピクチャ # ipicfrm 上のアイテムがサブモデルとなる。
org　サブモデル配置位置 (サブモデル原点の位置)
scf　サブモデル縮尺値
scf[0] : X 縮尺値 (サブモデルの X 軸方向に対する縮尺値) 負である場合には、サブモデルの Y 軸に対して反転する。
scf[1] : Y 縮尺値 (サブモデルの Y 軸方向に対する縮尺値) 負である場合には、サブモデルの X 軸に対して反転する。
拡大／縮小しないときは、scf[0] = scf[1] = 1.0 とする
ang　サブモデル回転角度 (単位 : 度)
Itmtyp　AwItem::SUBMODEL

【戻り値】

0 : 正常終了。異常終了の際は、サブモデルアイテムは作成されない。

注意
この関数は、指示された配置パラメータをもとに、テンポラリなサブモデルアイテムを追加する。
サブモデルアイテムを作るにはつぎのようにする。

```
AwTempItemDb* pTempItemDb = pModel->GetTempItemDb();
pTempItemDb->StoreItems();
if (Sub101(iname, keymsk, keydim, keydrf, ipicfrm, org, scf, ang, AwItem::SUBMODEL) == 0)
  pTempItemDb->StoreItems();
```

14.3.2 サブモデルアイテム作成

サブモデルアイテムを作成する。作成したアイテムはデータベース内に格納される。

【呼出し形式】

```
int Subput(const char* name, int mode, int keymsk, int keydim, int keydrf,
           int ipicfrm, const DPOINT& org, const double scf[], double ang,
           bool picRef = false)
```

【入力引数】

name　サブモデル名 (Null-char terminated)。サブモデル名に、モデルファイルの拡張子 ".MDL" を含める必要はない。
mode　サブモデル配置モード
0 : サブモデルとして配置する。
1 : 複合アイテムとして配置する。
2 : サブモデルを分解して配置する。
keymsk　アイテム選択マスク制御スイッチ
0 : 選択マスクを無視する。
1 : 一時的な選択マスクで指示されたアイテムだけをサブモデルアイテム化する。
keydim　寸法の調整制御スイッチ
0 : サブモデルに含まれる寸法を入力パラメータ (ORG, SCF, ANG) に合わせて調整しない。
1 : サブモデルに含まれる寸法を入力パラメータ (ORG, SCF, ANG) に合わせて調整する。
keydrf　製図図形要素の縮尺制御スイッチ

【戻り値】

0 : 正常終了。

14.4 アソシエイトアイテム

● 関数一覧

関数名	機能
Ascadd	アソシエイトアイテムにメンバーを追加する。
Ascbreak	アソシエイトの解除。
Ascdelete	アソシエイトアイテムとそのメンバーアイテムを削除する
Ascnameal	すべてのアソシエイトアイテムの名前を得る。
Ascnamegt	指定したアソシアエイトアイテムの名前を得る。
Ascnameid	与えた名前のアソシエイトアイテムのアイテム識別子を得る。
Ascnew	アソシエイトアイテムを作成する。
Ascrelese	あるアイテムをそれをメンバーとして含む全てのアソシエイトアイテムからはずす。
Ascrename	アソシエイトアイテムの名前を変更する。
Ascs29gt	アイテム識別子 idptr にアソシエイトされている idptr リストと、そのアソシエイトカテゴリ番号を取出す。
Ascctg101	カテゴリ番号の取り出し。
ascs29idp	アイテムがどのアソシエートアイテムに属するかを調べる。

14.4.1 アソシエイトアイテムにメンバー追加

アソシエイトアイテムにメンバーを追加する。

【呼出し形式】

```
int Ascadd(int idptrasc, const int idptrlist[], int ndptrcnt)
```

【入力引数】

idptrasc　アソシエイトアイテムのアイテム識別子
idptrlist　追加するアイテムの識別子リスト (int idptrlist[ndptrcnt])
ndptrcnt　追加アイテム数

【戻り値】

0　：正常終了
1　：idptrasc がアソシエイトアイテムでない

注意
追加アイテムに付けられるアソシエイトカテゴリ番号は idptrasc が持っているアソシエイトカテゴリ番号が追加される。

14.4.2 アソシエイトの解除

アソシエイトの解除。アソシエイトアイテムは削除されるが、そのアソシエイトアイテムのメンバーであったアイテムは削除されない。

【呼出し形式】

```
int Ascbreak(int idptrasc)
```

【入力引数】

idptrasc　アソシエイトアイテムのアイテム識別子

【戻り値】

0　：正常終了
1　：idptrasc がアソシエイトアイテムでない

14.4.3 アソシエイトアイテム及びメンバーアイテムをデータベースから削除

アソシエイトアイテムおよびそのメンバーであるアイテムを全てデータベースから削除する。

【呼出し形式】

```
int Ascdelete(int idptrasc)
```

【入力引数】

idptrasc　アソシエイトアイテムアイテム識別子

【戻り値】

0　：正常終了
1　：idptrasc がアソシエイトアイテムでない

14.4.4 全アソシエイトアイテム名を取得

すべてのアソシエイトアイテムの名前を得る。

【呼出し形式】

```
int Ascnameal(int icatasc, char itemname[][33])
```

【入力引数】

icatasc　アソシエイトカテゴリ番号
　0　：全てのアソシエイト名
　n　：アソシエイトカテゴリ番号 n を持つ全てのアソシエイトアイテムの名前

【出力引数】

itemname　アソシエイト名リスト (char itemname[][33])
　1 つの名前は 32 文字以下 (Null-char terminated)。

【戻り値】

名前の数 (≦ 4096)

14.4.5 指定したアソシエイトアイテムの名前を取得

指定したアソシアエイトアイテムの名前を得る。

【呼出し形式】

```
int Ascnamegt(int idptrasc, char *itemname)
```

【入力引数】

idptrasc　アイテム識別子

【出力引数】

itemname　アソシエイト名 (Null-char terminated)。名前は最大 32 文字。

【戻り値】

名前の文字数

14.4.6 アソシエイトアイテム名からアイテム識別子を取得

与えた名前のアソシエイトアイテムのアイテム識別子を得る。

【呼出し形式】

```
int Ascnameid(const char* itemname)
```

【入力引数】

itemname　アソシエイト名（Null-char terminated）。

【戻り値】

アイテム識別子。アソシエイトアイテムでなければ 0 を返します。

14.4.7 アソシエイトアイテムを作成

アソシエイトアイテムを作成する。

【呼出し形式】

```
int Ascnew(const char* itemname, int icatasc, const int idptrlist[], int ndptrcnt,
           int keyundo)
```

【入力引数】

itemname　アソシエイトアイテムの名前（Null-char terminated）。32 文字以下。
icatasc　アソシエイトカテゴリ番号 (1 - 255)
idptrlist　アソシエイトアイテムのメンバーにするアイテムの、アイテム識別番号のリスト。
(int idptrlist[ndptrcnt])
ndptrcnt　アイテム数
keyundo　UNDO をカウントアップするスイッチ
0　: カウントアップしない
1　: カウントアップする

【戻り値】

アソシエイトアイテムのアイテム識別子。エラーの場合 0 を返します。

14.4.8 アソシエイトアイテムからメンバーを外す

アソシエイトアイテムからメンバーをはずす。

【呼出し形式】

```
int Ascrelese(int idptrasc, const int idptrlist[], int idptrcnt)
```

【入力引数】

idptrasc　アソシエイトアイテムのアイテム識別子。
idptrlist　除去するアイテムのアイテム識別子。(int idptrlist[idptrcnt])
idptrcnt　除去するアイテム数。

【戻り値】

0　：正常終了
1　：このアイテムはどのアソシエイトアイテムのメンバーでもない

14.4.9 アソシエイトアイテム名を変更

アソシエイトアイテムの名前を変更する。

【呼出し形式】

```
int Ascrename(const char* itemname, int idptrasc)
```

【入力引数】

itemname　新しい名前（Null-char terminated)。32 文字以下。
idptrasc　アソシエイトアイテムのアイテム識別子

【戻り値】

0　：正常
1　：エラー

14.4.10 アイテム識別子のリストとカテゴリ番号の取得

アイテム識別子 idptr にアソシエイトされているアイテムのアイテム識別子のリストと、そのアソシエイトカテゴリ番号を取り出す。

【呼出し形式】

```
int Ascs29gt(int idptr, int icatasc, int items29[][2])
```

【入力引数】

idptr　アイテム識別子。アソシエイトアイテムまたはアソシエイトアイテムのメンバーであるアイテム。
icatasc　対象とするアソシエイトカテゴリ番号。
　0　: 全部のアソシエイトカテゴリを対象とする
　n　:n 番のアソシエイトカテゴリを対象とする

【出力引数】

items29　アソシエイト相手先データ (int items29[][2])
　items29[][0]　Ascctg101(items29[i][0], &major, &minor) 関数で major と minor の値を得る。
　　major : アソシエイトカテゴリ番号 (1-255)
　　minor : 親子関係
　　1 : items29[][1] がこのアイテムの親である

2 : items29[][1] がこのアイテムの子である
items29[][1]　　アソシエイト相手先アイテム識別子

【戻り値】

アソシエイト相手先数。アソシエイトされていなければ 0 を返します。

14.4.11 カテゴリ番号を取得

カテゴリ番号を取り出す。

【呼出し形式】

```
int Ascctg101(int category, int *category1, int *category2)
```

【入力引数】

category　カテゴリ番号

【出力引数】

category1　カテゴリ番号 1 格納領域
category2　カテゴリ番号 2 格納領域

【戻り値】

0　：正常
1　：エラー

14.4.12 アソシエートアイテムのアイテム識別子の取得

アイテムがどのアソシエートアイテムに属するかを調べる。

【呼出し形式】

```
int ascs29idp(int idptr, int icat)
```

【入力引数】

idptr　調べるアイテムのアイテム識別子
icat　アソシエートのカテゴリ番号
　0　：全部のアソシエイトカテゴリを対象とする。
　n　：n 番のアソシエイトカテゴリを対象とする。

【戻り値】

アソシエートアイテムのアイテム識別子
0　：アイテムはアソシエートされていない。

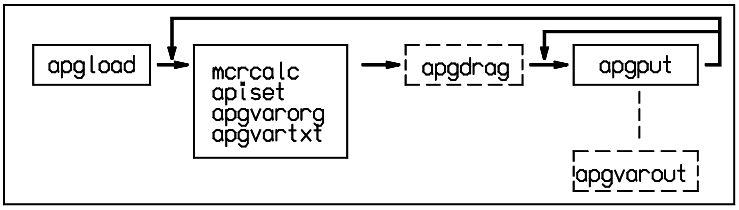
14.5 APG アイテム

APG ファイルは拡張子 (.APX) のテキストファイルでなければなりません。
APG Rev. 4 以上が対象です。

● 関数一覧

apgload　APG ファイルを APG データ領域にロードする。

apgput	**APG アイテム配置。**
apgvarorg	**APG ファイル作成時の値を 計算機変数にセットする。**
apgvartxt	**APG ファイル作成時のパラメータ名を取り出す。**
apgvarout	**APG パラメータ変数出力** **APG パラメータ名、およびその配置時の値をファイルに出力する。**
apgdrag	**APG ドラッグ。APG データをドラッグモードにする。**



14.5.1 APG ファイルの読み込み

APG ファイル を APG データ領域にロードする。一度ロードしたデータはつぎの APG ファイルがロードされるまで保持される。したがって、繰り返し位置を変えたりパラメータの値を変えたりして、連続して使用できる。

【呼出し形式】

```
int apgload(char *apgfile)
```

【入力引数】

apgfile　APG ファイル名 (Null-char terminated)。ファイル名に拡張子 ".APX" を含める必要はない。

【戻り値】

0　　: 正常終了

14.5.2 APG アイテム作成

APG データ領域にセットされた APG パラメータの値を、計算機変数中にセットされた値をもとに再計算します。そして、その値を使用して全図形を作成し、データベースに書き込みます。
apgput() 関数 は、クラス 251 で定義された寸法変数の値を計算し、計算機変数にその値をセットします。
apgput() の前に mcrcalc() 関数を使用して APG パラメータに値をセットしておきます。mcrcalc() 関数 については本書の「ユーティリティ」の章を参照して下さい。計算機変数とは、Advance CAD 中の計算 (コマンド中での計算式，マクロ中での外部変数，apgput() 時に使用された変数) で使用された変数のことです。APG パラメータ名は 6 文字以内でなければなりません。

【呼出し形式】

```
void apgput(int keyapg, int itemapg, int keyupd, int keydim, int keytol,
            const DPOINT& porg, const DPOINT& dvec, int newcls, int skp254,
            int clsofs, int outpic, int mirx, int miry, int keyspc,
            int keymsk, int keyuno)
```

【入力引数】

keyapg　Advance CAD データベースに作成するアイテムタイプの定義
　0　:APG ファイル作成時と同じアイテムタイプで作成される。
　1　: 全図形データが 1 つの APG アイテムとして作成される。
itemapg　既存の APG アイテムをアップデートする場合の APG アイテムの アイテム識別子。新規作成の場合 = 0
keyupd　必ず 0 をセットすること。
keydim　寸法線データ出力キー。
　0　: 寸法線を出力しない。1 : 寸法線を出力する。
keytol　パラメタライズ計算時に寸法公差を使用するためのスイッチ。
　0　: 使用しない。1 : 使用する。
porg　図形作成時の原点座標
dvec　図形作成時の回転ベクトル。たとえば、回転角度 30 度の場合は、
```
dvec->x = cos(30.0 * G2Math::ToRadian);
dvec->y = sin(30.0 * G2Math::ToRadian);
```
newcls　新規データ作成時のクラス番号
　0　:　APG ファイル作成時のクラス番号を使用する。
　1-255　:　データは、セットされたクラス番号で作成される。
skp254　APG 作成時にクラス 254 で作成されたデータをスキップするキー。
　0　:CLS 254 のデータをデータベースに書き込む。
　1　:CLS 254 のデータをデータベースに書き込まない。
clsofs　新規データの作成時に、APG ファイル作成時のクラス番号に加算するクラス番号。
クラス番号は、以下の式で決定される。
```
MAX( MOD(original class + clsofs), 256), 1)
```
outpic　出力ピクチャ番号をセットする (1 ～ AwItemAttributes::MAX_PICTURE)
　0　: カレントピクチャ番号
mirx　APG ファイル作成時の X 軸に対して反転してデータベースに書き込む。
　0　: 反転しない
　1　: 反転する
miry　APG ファイル作成時の Y 軸に対して反転してデータベースに書き込む。
　0　: 反転しない
　1　: 反転する
keyspc　APG アイテムを新規作成したときに、特性データバッファの内容を追加して、データベースに書き込むためのスイッチ。
　0　: 書き込まない。
　1　: 書き込む。
keymsk　アイテムタイプ、クラス、レビジョン、線種、線幅のマスク
　0　: 無効
　1　: 有効
keyuno　UNDO BLOCK をくぎるかどうか
　0　: くぎらない
　1　: くぎる

14.5.3 APG データをドラッグモードにする

APG データ領域にセットされた APG パラメータの値を計算器変数の値を使用して再計算する。そしてその値を使用して図形データを作成し、外形データ、最大ボックス、図形データそのままのいづれかをドラッグする。
apgdrag() 関数 を呼び出すと、クラス 251 で定義された寸法変数の値が計算され、計算器変数にその値が設定される。

【呼出し形式】

```
void apgdrag(int drgtyp, int keytol, const DPOINT& dvec, int skp254,
             int mirx, int miry, int keymsk)
```

【入力引数】

drgtyp　ドラッグモード。
- 1 : 外形データ
- 2 : 最大ボックス
- 3 : 図形データそのまま

keytol　パラメタライズ計算時に寸法公差を使用するためのスイッチ。
- 0 : 使用しない , 1 : 寸法公差を使用してパラメタライズ計算する。

dvec　図形作成時の回転ベクトル。たとえば回転角度 30 度の場合、

```
dvec.x = cos(30.0 * G2Math::ToRadian);
dvec.y = sin(30.0 * G2Math::ToRadian);
```

skp254　APG 作成時にクラス 254 で作成されたデータをスキップするかどうか。
- 0 : スキップしない
- 1 : スキップする

imirx　X 軸反転。
- 0 : 反転しない
- 1 : 反転する

imiry　Y 軸反転。
- 0 : 反転しない
- 1 : 反転する

keymsk　アイテムタイプ、クラス、レビジョン、線種、線幅のマスク。
- 0 : 無効
- 1 : 有効

14.5.4 APG ファイルパラメータ値を計算機変数に設定

APG ファイル作成時のパラメータの値を計算機変数にセットする

【呼出し形式】

```
void apgvarorg(void)
```

注意

APG ファイルをロードした直後に呼び出すこと。
apgdrag(), apgput() 関数のあとでこの関数を呼び出すと、配置またはドラッグの計算結果の値が計算機変数にセットされる。

14.5.5 APG ファイル作成時のパラメータ名を取得

APG ファイル作成時のパラメータ名を取り出す。

【呼出し形式】

```
int apgvartxt(int type, char nlst[][6], int maxlst)
```

【入力引数】

type　2
maxlst　配列 nlst の長さ

【出力引数】

nlst　パラメータ名を受ける配列 (char nlst[maxlst][6])

【戻り値】

パラメータ名の数

14.5.6 配置時の APG パラメータ名と値をファイルに出力

配置時の APG パラメータ名とその値をファイルに出力する。

【呼出し形式】

```
void apgvarout(const char* fname)
```

【入力引数】

fname　　出力ファイル名 (Null-char terminated)。ファイル名に拡張子 ".APP" を含める必要はない。

例

```
/
/ ** Advance CAD APG rev 4.0 Parameter Output **
/
AG1   = 50
H1    = 70
H2    = 50
R12   = 12
R30   = 30
R50   = 50
T     = 15
V1    = 45
V2    = 35
```

14.5.7 APG パラメータファイルのパラメータを計算機変数に設定

APG パラメータファイルのパラメータを、計算機変数にセットする。

【呼出し形式】

```
int apiset(const char* fname)
```

【入力引数】

fname　　APG パラメータファイル名 (Null-char terminated)。拡張子 ".API" を含める必要はない。

【戻り値】

0　　：正常終了

14.6 AwItemExploder クラス

このクラス はアイテムを分解するメソッドを持ちます。

14.6.1 コンストラクタ

```
AwItemExploder(AwModel* pModel);
```

AwModel オブジェクトへのポインタを渡します。

14.6.2 メソッド

コンポジットアイテムを分解します。コンポジットアイテムの構成要素をそれぞれ独立したアイテムとします。
int ExplodeComposits(AwItemListIterator* pListIterator, bool keepItem, bool changeAttr);
pListIterator　コンポジットアイテムを得る AwItemListIterator オブジェクトへのポインタ。
keepItem　元のコンポジットアイテムを残すかどうかを制御します。
false　削除する。
true　そのまま残す。
changeAttr　分解して出来るアイテムの属性の選択肢。
false　元のアイテムの属性を保持する。
true　コンポジットアイテムの表示モードに従って決める。
戻り値　コンポジットアイテムを分解して出来たパーマネントアイテムの数を返します。

製図アイテムを分解しコンポジットアイテムにします。
１つのアイテムを分解してできるアイテムすべてを構成要素とするコンポジットアイテムを作ります。
元の製図アイテムは削除します。
int ExplodeDrafs(AwItemListIterator* pListIterator);
pListIterator　製図アイテムを得る AwItemListIterator オブジェクトへのポインタ。このメソッドが処理するアイテムの種類は AwItem::TEXT、MARK、DIMENSION、GTOL、XHT です。
戻り値　製図アイテムを分解して出来たパーマネントアイテムの数を返します。

製図アイテム内の対象サブレコードシーケンスは下記のものです。
AwSr::POINT
　点アイテムに変換します。
AwSr::STARTPOINT, { AwSr::LINE, AwSr::CIRCLE, AwSr::BEZIER }+
　ストリングアイテムに変換します。
AwSr::TEXT
　１文字を１つのストリングアイテムに変換します。
　一筆画きにするために不連続なパスは非表示の線分で連結します。
AwSr::MARK
　スナップノードは点アイテムに変換します。
　それ以外は１つのストリングアイテムに変換します。
　一筆画きにするために不連続なパスは非表示の線分で連結します。

塗り潰しアイテムを分解します。
int ExplodeAfls(AwItemListIterator* pListIterator, bool keepItem);
pListIterator　塗り潰しアイテムを得る AwItemListIterator オブジェクトへのポインタ。
keepItem　元の塗り潰しアイテムを残すかどうかを制御します。
false　削除する。
true　そのまま残す。
戻り値　塗り潰しアイテムを分解して出来たパーマネントアイテムの数を返します。

曲線アイテムを分解します。
int ExplodeCurves(AwItemListIterator* pListIterator);
pListIterator　曲線アイテムを得る AwItemListIterator オブジェクトへのポインタ。このメソッドが処理するアイテムの種類は AwItem::LINE、CIRCLE、SPLINE、STRING です。
戻り値　曲線アイテムを分解して出来たパーマネントアイテムの数を返します。

１つの曲線アイテムは分解するといくつかの線分アイテム、円弧アイテム、スプラインアイテムになります。
曲線アイテムの中の幾何図形それぞれを１つの曲線アイテムに変換します。
3 次 Bezier 曲線が連続する場合は１つのスプラインアイテムに変換します。連続していてもサブレコードの線種が変わるときは、そこまででひとつのスプラインアイテムにします。

分解してできる曲線アイテムの線種は元の曲線サブレコードの線種です。
スプラインのトリムされている部分、曲線アイテムの非表示部分はアイテムとはせず取り除きます。
元の曲線アイテムは削除します。

14.7 AwItemEditor クラス

このクラスはパーマネントアイテムに対して下記の操作を行います。
アイテムの移動、回転、反転などの編集操作をする。
- **複製**
- **平行移動**
- **回転移動**
- **平行移動と回転移動**
- **反転**
- **スケーリング（拡大または縮小）**
- **ストレッチ**

アイテムの配列を作る。
- **格子状配列**
- **円形配列**
- **向心円形配列**

14.7.1 コンストラクタ

```
AwItemEditor(AwModel* pModel);
```
AwModel オブジェクトへのポインタを渡します。

14.7.2 条件

指定したアイテムを編集するか,新しくアイテムを作成するかどうかを設定します。
```
bool SetTargetItem(int type);
```
type		処理方法を示す整数。
	0	パーマネントアイテムを編集する。(デフォルト)。
	1	元のアイテムの属性を持つ新しいアイテムを作る。
	2	「現在のアイテム属性」を持つ新しいアイテムを作る。
戻り値		設定できたとき true を返します。

アイテムのピクチャ番号を設定します。
```
bool SetTargetPicture(int pid);
```
pid	アイテムのピクチャ番号 (1-)。元のアイテムと同じピクチャ番号にしたければ特別な値 0 を設定します。設定しない場合のデフォルト値は 0 です。
戻り値	設定できたとき true を返します。

指定したアイテムを編集するか、指定したアイテムをもとに新しくアイテムを作成するかを指定できます。またアイテムのピクチャ番号を指定できます。これらは、編集を行う前に指定します。
下記のコードフラグメントは、アイテムを X 方向に 10 移動した新しいアイテムをピクチャ 12 に作成します。元のアイテムは変更しません。
```
AwItemEditor itemEditor(pModel);
itemEditor.SetTargetItem(1);       // 新しいアイテム
itemEditor.SetTargetPicture(12);  // ピクチャ 12
if (! itemEditor.TranslateItems(&iterator, G2Point(10.0, 0.0)))
  return; // ERROR
```

14.7.3 アイテム移動・回転・反転

アイテムの複製を作成します。

```
bool CopyItems(AwItemListIterator* pIterator);
```
pIterator　パーマネントアイテムを得る AwItemListIterator オブジェクトへのポインタ。
戻り値　成功したとき true を返します。

アイテムを平行移動します。
```
bool TranslateItems(AwItemListIterator* pIterator, const G2Point& vec);
```
pIterator　パーマネントアイテムを得る AwItemListIterator オブジェクトへのポインタ。
vec　移動量。
戻り値　成功したとき true を返します。

アイテムを回転移動します。
```
bool RotateItems(AwItemListIterator* pIterator, const G2Point& origin, double angle);
```
pIterator　パーマネントアイテムを得る AwItemListIterator オブジェクトへのポインタ。
origin　回転中心。
angle　回転角度（度）。
戻り値　成功したとき true を返します。

アイテムを平行移動と回転移動します。
```
bool TransformItems(AwItemListIterator* pIterator, const G2Point& vec, const G2Point& origin,
                    double angle);
```
pIterator　パーマネントアイテムを得る AwItemListIterator オブジェクトへのポインタ。
vec　移動量。
origin　回転中心。
angle　回転角度（度）。
戻り値　成功したとき true を返します。

アイテムを指定した軸に対して反転します。
```
bool MirrorItems(AwItemListIterator* pIterator, const G2Point& origin, const G2Vector& uvec);
```
pIterator　パーマネントアイテムを得る AwItemListIterator オブジェクトへのポインタ。
origin　反転軸の通過点。
uvec　反転軸の向き。
戻り値　成功したとき true を返します。

アイテムをスケーリング（拡大または縮小）します。
```
bool ExpandItems(int keydrf, AwItemListIterator* pIterator, const G2Point& origin,
                 const double scale[]);
```
keydrf　テキストアイテム、マークアイテムも縮尺するかどうかの制御スイッチ。
　0　縮尺しない。
　1　縮尺する。
pIterator　パーマネントアイテムを得る AwItemListIterator オブジェクトへのポインタ。
origin　スケーリング中心点。
scale　倍率 (scale[0] != 0.0, scale[1] != 0.0)。
　scale[0] : X 縮尺値。負の場合は Y 軸に対して反転する。
　scale[1] : Y 縮尺値。負の場合は X 軸に対して反転する。
戻り値　成功したとき true を返します。

アイテムをストレッチします。アイテムのストレッチ領域内に入る部分だけを移動します。アイテム全体がストレッチ領域内に入っていれば平行移動します。
```
bool StretchItems(AwItemListIterator* pIterator, const G2Point& vec);
```
pIterator　パーマネントアイテムを得る AwItemListIterator オブジェクトへのポインタ。
vec　移動量。
戻り値　成功したとき true を返します。

14.7.4 アイテムの配列

アイテムの格子状配列を作ります。

```
bool RectArray(AwItemListIterator* pIterator, const short nele[], const G2Point vec[]);
```

pIterator	パーマネントアイテムを得る AwItemListIterator オブジェクトへのポインタ。
nele	行と列方向の要素数 (1 <= nele[0], nele[1])。
vec	行と列方向の移動量。
戻り値	成功したとき true を返します。

アイテムの円形配列を作ります。

```
bool CircularArray(AwItemListIterator* pIterator, int nele, const G2Point& origin,
                   double angle, const G2Point& bse);
```

pIterator	パーマネントアイテムを得る AwItemListIterator オブジェクトへのポインタ。
nele	円周上に作る要素数 (1 <= nele)。
origin	円中心点。
angle	要素間角度（度）。
bse	アイテムの基準位置。
戻り値	成功したとき true を返します。

アイテムの向心円形配列を作ります。

```
bool RadialArray(AwItemListIterator* pIterator, int nele, const G2Point& origin,
                 double angle);
```

pIterator	パーマネントアイテムを得る AwItemListIterator オブジェクトへのポインタ。
nele	円周上に作る要素数 (1 <= nele)。
origin	円中心点。
angle	要素間角度（度）。
戻り値	成功したとき true を返します。

14.7.5 edtstrset() 関数

ストレッチ領域を設定します。ストレッチ領域はアクティブモデルの AwEditParam オブジェクトが保持します。

【呼出し形式】

```
int edtstrset(const FPOINT plypnt[], int npnt)
```

【入力引数】

plypnt	多角形の伸縮領域 (FPOINT plypnt[npnt]) npnt が 2 の場合は、長方形の伸縮領域の対角点とみなす。
npnt	多角形の伸縮領域のコーナーの総数 (2 <= npnt <= 256)。

第 15 章　特性データ

15.1 特性データ

● 関数一覧

Spc005	**アイテムに付加された特性データを取り出す。**
Spc101	**カレントの特性データのファイル番号とレコード番号を変更する。**
Spc102	**カレントの特性データにデータをセットする。**
Spc103	**カレントの特性データをアイテムに追加する。**
Spc104	**カレントの特性データをアイテムから削除する。**

15.1.1 アイテムに付加された特性データを取得

アイテムに付加された特性データを取り出す

【呼出し形式】

```
int Spc005(int idptr, int rec, int num, double *dval, int *c_dval,  char *cval,
          size_t l_cval)
```

【入力引数】

idptr	アイテム識別番号
rec	特性データのレコード番号
num	特性データの項目番号
c_dval	dval の配列の個数
l_cval	cval の大きさ（バイト単位）

【出力引数】

dval	付加された特性データ（数値の場合）
c_dval	dval に格納した特性データの個数（数値の場合）
cval	付加された特性データ（文字列の場合。Null-char terminated）

【戻り値】

0	: 文字列。
1	: 16 ビット整数。
2	: 単精度実数。
3	: 倍精度実数。
-1	: アイテム識別子が正しくない。
-2	: アイテムには特性データが付加されていない。
-3	: 指定されたレコード番号の特性データが付加されていない。
-4	: 指定された項目番号の特性データが付加されていない。
-5	: 特性データのレコード番号が正しくない。

-6　　：特性データの項目番号が正しくない。

注意

取り出される特性データが数値（16 ビット整数、単精度実数または倍精度実数）の場合、数値は倍精度実数に変換されて、配列 dval に格納される。c_dval には用意した配列 dval の個数を指定する。c_dval には取り出した特性データの個数が設定される。取り出される特性データが文字列の場合、cval に格納される。取り出された文字列は Null-char terminated となるので、少なくとも、取り出される文字列の長さに 1 を足した領域を、cval として用意する必要がある。l_cval には、用意した領域の大きさをバイト単位で指定する。

15.1.2 カレント特性データのファイル番号とレコード番号を変更

カレントの特性データのファイル番号とレコード番号を変更する。

【呼出し形式】

```
int Spc101(int fleno, int recno, int keyint)
```

【入力引数】

fleno　特性データファイル番号
recno　特性データレコード番号
keyint　特性データ初期化スイッチ
　　0　　：カレントの特性データを初期化しない。
　　1　　：カレントの特性データを初期化する。

【戻り値】

0　　：正常終了

15.1.3 カレント特性データにデータを設定

カレントの特性データにデータをセットする。

【呼出し形式】

```
int Spc102(int kno, int type, int cnt, const char* text, const double slst[])
```

【入力引数】

kno　　特性データの項目番号
type　　入力データのタイプ。
　　0　　：文字列でデータをセットする
　　1　　：数値をセットする。
cnt　　入力データの数。
　　type == 0 のとき文字列の長さ、type == 1 のとき数値の数。
text　　文字列
slst　　数値の並び

【戻り値】

0　　：正常終了

15.1.4 カレント特性データをアイテムに追加

カレントの特性データをアイテムに追加する。

【呼出し形式】

```
int Spc103(int idptr)
```

【入力引数】

idptr　　アイテムの識別番号

【戻り値】

0　　：正常終了

15.1.5 カレント特性データをアイテムから削除

カレントの特性データをアイテムから削除する。

【呼出し形式】

```
int Spc104(int idptr)
```

【入力引数】

idptr　　アイテム識別番号

【戻り値】

0　　：正常終了
-1　　：アイテム識別番号が正しくない
-2　　：アイテムには特性データが付加されていない。

第 16 章　シンボルとモデルファイル

16.1 シンボルファイル

● 関数一覧

関数名	機能
SymGenI	シンボルファイルを作成する
SymFrfHdrI	シンボルファイルのヘッダー情報を参照する。
SymDspWinI	シンボルを指定ウインドウ領域内に表示する。
SymFrfNdpI	シンボルファイルのノードポイントを参照する。

16.1.1 シンボルファイルを作成

シンボルファイルを作成する。

【呼出し形式】

```
int SymGenI(const char* name, int keygen, const DPOINT* org, const int ibstxt[],
            int iasclst[][5], int nasc)
```

【入力引数】

name　シンボル名 (Null-char terminated)。シンボル名に、シンボルファイルの拡張子 ".SYM" を含める必要はない。

keygen　シンボルファイルの作成モード

- 0 : アクティブリストのアイテムからシンボルファイルを作成する。
- ≧ 1 : モデルのピクチャ #keygen 上のアイテムからシンボルファイルを作成する。

org　シンボル原点

ibstxt　ベーステキストの指定 (int ibstxt[2])

- ibstxt[0] : ベーステキストとするテキストセグメントを持つアイテムのアイテム識別子
- ibstxt[1] : テキストセグメントの直前の分類サブレコードの相対番号。
  シンボルにベーステキストを持たせない場合は、ibstxt[0] と ibstxt[1] を 0 とする。

iasclst　ノードテキストの指定リスト (int iasclst[nasc][5])

- iasclst[][0] : ノードテキストとするテキストセグメントを持つアイテムのアイテム識別子

iasclst[][1] : テキストセグメントの直前の分類サブレコードの相対番号
iasclst[][2] : ノードテキストを付加するポイントセグメントを持つアイテムのアイテム識別子
iasclst[][3] : ポイントセグメントの点サブレコードの相対番号
iasclst[][4] : ノードテキストの分類番号 (分類番号 :1 ～ 3)

nasc　ノードテキストの指定の総数。ノードポイントにノードテキストを付加しない場合は、nasc を 0 とする。ノードテキストの指定の総数 nasc の最大値は 768 である。

【戻り値】

0 : 正常終了

注意

シンボルファイルの作成の対象となるアイテムに含まれる非表示のポイントセグメントは、すべてシンボルのノードポイントとなる。シンボルのノードポイントの最大数は 255。これを越えるポイントセグメントがある場合は、シンボルファイルは作成されない。

以下の場合はシンボルファイルは作成されない。

- **シンボルファイルに含めるアイテムが持つサブレコードの総数が 16384 を越える場合**
- **サブレコードデータのデータ量が 256 Kword を越える場合**

ベーステキストおよびノードテキストとするテキストセグメントを持つアイテムは、シンボルファイルの作成の対象となるアイテムに含まれていなければならない。
同一のテキストセグメントをベーステキストと複数のノードポイントのノードテキストとすることはできない。
通常、グラフィックステキストアイテムが 1 つのテキストセグメントだけからなる場合、テキストセグメントの直前の分類サブレコードの相対番号は 1 になる。
また、点アイテムが 1 つのポイントセグメントだけからなる場合、ポイントセグメントの点サブレコードの相対番号は 1 になる。
IdentItem 関数 でアイテムをピックした場合、ピックされたアイテムのアイテム識別番号、テキストセグメントの直前の分類サブレコードの相対番号およびポイントセグメントの点サブレコードの相対番号は、IdentInfo 関数で得ることができる。

16.1.2 シンボルファイルのヘッダー情報を参照

シンボルファイルのヘッダー情報を参照する。

【呼出し形式】

```
int SymFrfHdrl(const char* name, short icdate[], char* cusrnme, DPOINT box[])
```

【入力引数】

name　シンボル名 (Null-char terminated)。シンボル名に、シンボルファイルの拡張子 ".SYM" を含める必要はない。

【出力引数】

idate　シンボルファイル作成日時 (short idate[6])
idate[0] : 日、idate[1] : 月、idate[2] : 年
idate[3] : 時、idate[4] : 分、idate[5] : 秒

cusrnme　シンボルファイルのユーザ名 (char cusrnme[17], Null-char terminated)

box　シンボルボックス (DPOINT box[4])
box[0] : シンボルボックスの左下の点
box[1] : シンボルボックスの右下の点
box[2] : シンボルボックスの左上の点
box[3] : シンボルボックスの右上の点

【戻り値】

0　　：正常終了

16.1.3 シンボルをウィンドウに表示

シンボルを指定ウインドウ領域内に表示する。

【呼出し形式】

```
int SymDspWinl(const char* name, int keyscr, int keyers, int iwindsp, const short iwinzon[]);
```

【入力引数】

name　　シンボル名 (Null-char terminated)。シンボル名にシンボルファイルの拡張子 ".SYM" を含める必要はない。

keyscr　　表示プレーン制御スイッチ

- 0　　: 単色表示。テンポラリ図形用プレーンにシンボルを表示する。
- 1　　: カラー表示。実図形用プレーンにシンボルを表示する。

keyers　　ウインドウ領域消去スイッチ

- 0　　: ウインドウ領域内を消去しないで、シンボルを表示する。
- 1　　: ウインドウ領域内を消去して、シンボルを表示する。

iwindsp　　表示ウインドウ番号

- 1　　: グラフィックウインドウ
- 2　　: サブグラフィックウインドウ (ＭＳＣ エリア)

iwinzon　　表示ウインドウ領域

- iwinzon[0]　　:　ウインドウ領域の左下 X 座標 (ラスター)
- iwinzon[1]　　:　ウインドウ領域の左下 Y 座標 (ラスター)
- iwinzon[2]　　:　ウインドウ領域の右上 X 座標 (ラスター)
- iwinzon[3]　　:　ウインドウ領域の右上 Y 座標 (ラスター)

【戻り値】

0　　：正常終了

注意

以下はサブグラフィックウインドウ (MSC エリア) にシンボルを表示するときの例。

```
short iwinzon[16];
int iwinno = 2;
Menuzone(2, iwinno, iwinzon);
SymDspWinl(iname, keydsp, keyers, iwinno, iwinzon);
```

16.1.4 シンボルファイルのノードポイントを参照

シンボルファイルのノードポイントを参照する。

【呼出し形式】

```
int SymFrfNdpl(const char* name, short* ndpnt, DPOINT pnts[]);
```

【入力引数】

name　　シンボル名 (Null-char terminated)。シンボル名に、シンボルファイルの拡張子 ".SYM" を含める必要はない。

【出力引数】

ndpnt　　シンボルファイルのノードポイント数 (0 ≦ NDPNT ≦ 255)
pnts　　シンボルファイルのノードポイントの座標 (DPOINT pnt[256])

【戻り値】

0　　: 正常終了

注意
シンボルファイルのノードポイントの座標は、シンボルの原点を (0,0) としたときの座標である。

16.2 モデルファイル

● 関数一覧

関数名	機能
Mdlinit	アクティブモデルを初期化する。
Mdlwrite	アクティブモデルをモデルファイルに保存する。
Mdlread	既存のモデルファイルを読み込む。
Mdlread1	既存のモデルファイルを読み込む。
Mdlrplpic	アクティブモデルのピクチャ書き込みを行なう。
Mdl104	モデルファイルのモデルタイトルを参照する。
Mdl105	モデルファイルのヘッダー情報を参照する。
Mdlname001	アクティブモデル名を設定／解除する。
Mdlname101	アクティブモデル名を得る。
Subgen	サブモデルファイルを作成する。

16.2.1 モデルの初期化

アクティブモデルを初期化する。

【呼出し形式】

```
int Mdlinit(void)
```

【戻り値】

0　　: 正常終了

16.2.2 アクティブモデルをファイルに保存

アクティブモデルをモデルファイルに保存する。

【呼出し形式】

```
int Mdlwrite(char* name, int mode, int keychk, int keyinf,
             int keypcvt, const short ipcvtbl[], bool picRef = false)
```

【入力引数】

name　　モデル名 (Null-char terminated)。
　　モデル名にモデルファイルの拡張子 ".MDL" を含める必要はない。
mode　　モデルの保存モード
　　-1　　: アクティブリスト中のアイテムだけを保存する。アクティブリストのアイテムがあるピクチャについてだけピクチャ属性を保存する。
　　0　　: モデルの全てのピクチャを保存する。
　　≧ 1　　: 保存するピクチャの番号。
keychk　　作成モデルファイル検証スイッチ
　　0　　: 検証しない
　　1　　: サイズを検証
keyinf　　モデル情報ファイルの出力制御スイッチ
　　0　　: モデル情報ファイルを出力しない。
　　1　　: モデル情報ファイルを出力する。
　　2　　: モデル情報ファイルを更新出力する。
keypcvt　　ピクチャ番号変換スイッチ
　　0　　: ピクチャ番号を変換しない。
　　1　　: ピクチャ番号を変換する。
ipcvtbl　　ピクチャ番号変換テーブル
picRef　　ピクチャ参照機能の制御スイッチ
　　(ピクチャ参照についてはコマンドリファレンスマニュアルを参照)
　　false　　: ピクチャ参照機能を使用しない。
　　true　　: ピクチャ参照機能を使用する。

【戻り値】

0　　: 正常終了

注意
　モデルの保存モード mode が 0 のとき、アクティブモデルのモデル名が name で指定された名前となる。

16.2.3 モデルファイルの読み込み

既存のモデルファイルを読み込む。

【呼出し形式】

```
int Mdlread(const char* name, int mode, int keymsk, int ipicfrm, int ipicdst,
            double ang, bool picRef = false)
```

【入力引数】

name　　モデル名 (Null-char terminated)。
　　モデル名にモデルファイルの拡張子 ".MDL" を含める必要はない。
mode　　モデルの読み込みモード
　　0　　: 新規開始してからモデルを読み込む。
　　1　　: 現在作業中のモデルにアイテムだけを読み込む。
　　2　　: 現在作業中のモデルにアイテムを読み込み、ピクチャの縮尺値をモデルファイルと同じにする。
keymsk　　選択マスクの制御スイッチ
　　0　　: 現在の選択マスクを参照せずに、全てのアイテムを読み込む。
　　1　　: 現在のアイテム選択マスクとクラス選択マスクに従って、アイテムを読み込む。
ipicfrm　　入力ピクチャ番号
　　0　　: 全てのピクチャのデータを読み込む。
　　> 0　　: ピクチャ #ipicfrm のデータだけ読み込む。

ipicdst　格納先ピクチャ番号
　0　: アイテムを各々のピクチャに読み込む。
　> 0　: アイテムをピクチャ #ipicdst に読み込む。
ang　読み込み時の回転角度
picRef　ピクチャ参照機能の制御スイッチ
（ピクチャ参照についてはコマンドリファレンスマニュアルを参照）
　false　: ピクチャ参照機能を使用しない。
　true　: ピクチャ参照機能を使用する。

【戻り値】

0　: 正常終了

注意
入力引数の値が以下のとき、アクティブモデルのモデル名が name で指定された名前となる。

モデルの読み込みモード	mode == 0
選択マスクの制御スイッチ	keymsk == 0
入力ピクチャ番号	ipicfrm == 0
格納先ピクチャ番号	ipicdst == 0

16.2.4 モデルファイルの読み込み

既存のモデルファイルを読み込む。

【呼出し形式】

```
int Mdlread1(const char* name, int mode, int picscf, int romode, int keymsk,
             const short ipcvtbl[], double ang, bool picRef = false)
```

【入力引数】
name　モデル名 (Null-char terminated)。
モデル名にモデルファイルの拡張子 ".MDL" を含める必要はない。
mode　モデルの読み込みモード
　0　: 新規開始してからモデルを読み込む。
　1　: 現在作業中のモデルにアイテムだけを読み込む。
picscf　picture scale 制御 (常に 0 を渡すこと)
　0　: ピクチャの縮尺値は変更しない。
　1　: ピクチャの縮尺値をモデルファイルと同じにする。
romode　読み込み専用モード
　0　: 通常モード
　1　: 読み込み専用モードでモデルファイルを読み込む。
keymsk　選択マスクの制御スイッチ
　0　: 現在の選択マスクを参照せずに、全てのアイテムを読み込む。
　1　: 現在のアイテム選択マスクとクラス選択マスクに従って、アイテムを読み込む。
ipcvtbl　ピクチャ変換テーブル (short ipcvtbl[AwItemAttributes::MAX_PICTURE])
ipcvtbl[i - 1]:ピクチャ #i をピクチャ #ipcvtbl[i - 1] に変換。
(i == 1, AwItemAttributes::MAX_PICTURE)
ipcvtbl[i] の値は、1 から AwItemAttributes::MAX_PICTURE までで、かつ ipcvtbl[j] (j<>i) と重複してはならない。
ang　読み込み時の回転角度
picRef　ピクチャ参照機能の制御スイッチ
（ピクチャ参照についてはコマンドリファレンスマニュアルを参照）
　false　: ピクチャ参照機能を使用しない。
　true　: ピクチャ参照機能を使用する。

【戻り値】

0　　：正常終了

注意

入力引数の値が以下のとき、アクティブモデルのモデル名が name で指定された名前となる。

モデルの読み込みモード	mode　== 0
選択マスクの制御スイッチ	keymsk == 0
ピクチャ変換テーブル	ピクチャ変換を全くしないとき

16.2.5 モデルピクチャ書き込み

アクティブモデルの指定ピクチャだけをモデルファイルに書き込む。

【呼出し形式】

```
int Mdlrplpic(const char* name, int keychk, int ipicmdf, int ipicmdl,
              bool picRef = false)
```

【入力引数】

name	モデル名 (Null-char terminated)。 モデル名にモデルファイルの拡張子 ".MDL" を含める必要はない。 既存のモデルファイルでなければならない。
keychk	使用しない引数（0 を渡せばよい)
ipicmdf	書き込み先ピクチャ モデルファイルのピクチャ番号
ipicmdl	書き込み元ピクチャ モデルのピクチャ番号
picRef	ピクチャ参照機能の制御スイッチ （ピクチャ参照についてはコマンドリファレンスマニュアルを参照） false　：ピクチャ参照機能を使用しない。 true　：ピクチャ参照機能を使用する。

【戻り値】

0　　：正常終了

16.2.6 モデルファイルのモデルタイトルを参照

モデルファイルのモデルタイトルを参照する。

【呼出し形式】

```
int Mdl104(const char* name, short *ittllst, int nttl, MDLTTLTXT *cttltxt,
           short *lttltxt)
```

【入力引数】

name	モデル名 (Null-char terminated)。モデル名にモデルファイルの拡張子 ".MDL" を含める必要はない。
ittllst	参照するモデルタイトル番号の配列 (short ittllst[nttl])
nttl	参照するモデルタイトル番号の総数

【出力引数】

cttltxt	モデルタイトルの配列 (MDLTTLTXT cttltxt[nttl], Null-char terminated)。

lttltxt　　モデルタイトルの文字数 (short lttltxt[nttl])

【戻り値】

0　　：正常終了

16.2.7 モデルファイルのヘッダー情報参照

モデルファイルのヘッダー情報を参照する。

【呼出し形式】

```
int Mdl105(const char* name, short cdate[], short udate[], char* username,
           short* unit, char* mdlttl, MDLPICNME picnames[],
           int* nttlitm, int npicitm[], int* nascitm, bool display)
```

【入力引数】

name　　モデル名 (Null-char terminated)。モデル名にモデルファイルの拡張子 ".MDL" を含める必要はない。

【出力引数】

cdate　　モデルファイル作成日時 (short icdate[6])
cdate[0] : 日，cdate[1] : 月，cdate[2] : 西暦年
cdate[3] : 時，cdate[4] : 分，cdate[5] : 秒

udate　　モデルファイル更新日時 (short iudate[6])
udata[0] : 日，udata[1] : 月，udata[2] : 西暦年
udata[3] : 時，udata[4] : 分，udata[5] : 秒

username　　モデルファイルのユーザ名 (char username[17], Null-char terminated)。

unit　　モデルの長さの単位
- 1　：ミリ単位系
- 4　：インチ単位系

mdlttl　　モデル主タイトル (Null-char terminated)。

picnames　　ピクチャ名の配列 (MDLPICNME picnames[AwItemAttributes::MAX_PICTURE], Null-char terminated)。
ピクチャ 1 からピクチャ AwItemAttributes::MAX_PICTURE までのピクチャ名。

nttlitm　　総アイテム数。

npicitm　　ピクチャ毎のアイテム数 (int npicitm[AwItemAttributes::MAX_PICTURE])

nascitm　　アソシエーションアイテム数。

display　　モデルファイルの情報をスクリーンに表示するかどうか。
- true　：表示する。
- false　：表示しない。

【戻り値】

0　　：正常終了

16.2.8 アクティブモデル名を設定または解除

アクティブモデル名を設定または解除する。具体的には以下の２つの処理を行う。

(1)　**モデルタイトル項目番号２０２を設定／解除。**
(2)　**モデルのロックファイルの作成／解除。**

【呼出し形式】

```
int Mdlname001(const char* mdlnme)
```

【入力引数】

mdlnme　モデル名 (Null-char terminated)。
　　解除する場合は (const char *)NULL または空文字列（""）にする。

【戻り値】

0　：正常終了。
-n　：正常であるが指定されたモデル名は以下の状態にある。
　-50　：指定されたモデルが既に存在する。
　-51　：書き込み禁止のファイル／ディレクトリが指定された。
　　または呼出し専用オプション（-ro）で起動された。
　-52　：指定されたモデルは他のユーザによって使用されている。
+n　：指定されたモデル名がモデルタイトル２０２として不正。
　2　：モデルタイトルテンプレートで規定されている最小文字数より短い。
　3　：モデルタイトルテンプレートで規定されている最大文字数より長い。
　4　：モデルタイトルの総文字数が制限を超える。
　5　：モデルタイトルテンプレートで規定されている使用禁止文字が含まれている。

例

```
int ier;
ier = Mdlname001("TEST");              /* モデル名を TEST にする */
ier = Mdlname001((const char *)NULL);  /* モデル名を解除する */
```

16.2.9 アクティブモデル名を得る

アクティブモデル名を得る。

【呼出し形式】

```
int Mdlname101(int iswt, char* mdlnme, size_t l_mdlnme)
```

【入力引数】

iswt　スイッチ。
　1　：入力された状態のモデル名を得る。
　2　：フルパスのモデル名を得る。
　3　：ディレクトリとファイル拡張子を取り除いたモデル名を得る。
l_mdlnme　出力引数 mdlnme の大きさ（バイト数）。

【出力引数】

mdlnme　モデル名 (Null-char terminated)。

【戻り値】

0　：正常終了。
1　：入力引数 iswt の値が不正。
2　：モデル名が l_mdlnme 以上。
3　：モデル名は設定されていない。

例

```
int ier;
char mdlnme1[257], mdlnme2[257], mdlnme3[257];
(void)Mdlname001("test");
ier = Mdlname101(1, mdlnme1, sizeof(mdlnme1));
```

```
ier = Mdlname101(2, mdlnme1, sizeof(mdlnme2));
ier = Mdlname101(3, mdlnme1, sizeof(mdlnme3));

mdlnme1、mdlnme2、mdlnme3 は以下のような文字列になる。
mdlnme1 : "test"
mdlnme2 : "/acad/files/TEST.MDL"
mdlnme3 : "TEST"
```

16.2.10 サブモデルファイルの作成

サブモデルファイルを作成する。

【呼出し形式】

```
int Subgen(const char* name, int mode, int keychk, int keyrpl, int keyinf,
          int keypcvt, const short ipcvtbl[],
          const DPOINT org[], const double ang[])
```

【入力引数】

name　サブモデル名 (Nll-char terminated)。サブモデル名に、モデルファイルの拡張子 ".MDL" を含める必要はない。

mode　サブモデルファイルの作成モード
- -1　: アクティブリスト中のアイテムだけをサブモデルのアイテムとする。
- 0　: 全アイテムをサブモデルのアイテムとする。
- ≧ 1　: モデルのピクチャ #mode 上のアイテムをサブモデルのアイテムとする。

keychk　作成サブモデルファイル検証スイッチ
- 0　: 検証しない
- 1　: サイズを検証

keyrpl　サブモデル変換制御スイッチ
- 0　: サブモデルのアイテムとしたアイテムをサブモデルアイテムに変換しない。
- 1　: サブモデルのアイテムとしたアイテムをサブモデルアイテムに変換する。

keyinf　モデル情報ファイルの出力制御スイッチ
- 0　: モデル情報ファイルを出力しない。
- 1　: モデル情報ファイルを出力する。
- 2　: モデル情報ファイルを更新出力する。

keypcvt　ピクチャ変換制御スイッチ
- 0　: ピクチャ変換しない。
- 1　: アイテムのピクチャ番号とピクチャ縮尺値をピクチャ変換テーブルにもとづき変換する。

ipcvtbl　ピクチャ変換テーブル (short ipcvtbl[AwItemAttributes::MAX_PICTURE])
ピクチャ変換制御スイッチ KEYPCVT == 1 のときだけ参照される。
ipcvtbl[i - 1]　:　ピクチャ #i をピクチャ #ipcvtbl[i - 1] に変換。(i == 1, AwItemAttributes::MAX_PICTURE)
ipcvtbl[i] の値は、1 から AwItemAttributes::MAX_PICTURE までで、かつ ipcvtbl[j] (j<>i) と重複してはならない。

org　サブモデル原点の配列 (DPOINT org[AwItemAttributes::MAX_PICTURE])

ang　サブモデル基準水平角度の配列 (単位 : 度)(double ang[AwItemAttributes::MAX_PICTURE])

【戻り値】

0　:　正常終了

第 17 章　ユーザ定義の定数

この章ではユーザ定義の定数を利用する方法とモデル管理をカスタマイズする方法を説明します。アプリケーションはこれらの機能を実現するためいくつかのフック関数を呼び出します。いつどんなフック関数を呼び出すかは後で説明します。フック関数のデフォルトの実装は「何もしない」関数です。ユーザが独自の処理を追加できるように入り口を用意しているだけです。このようなユーザによる変更を許す部分をフレームワークのホットスポットといいます。

17.1 ユーザ定義の定数

ユーザ定義の定数を設定することができます。ユーザ定義定数はユーザコマンドを追加することで会話形式で値の表示／変更ができます。
またユーザ定義定数はモデルファイルの一部として保存することができます。

以下では、ユーザ定義定数に必要なメニューファイル、実装すべき関数について説明します。メニューファイル、ソースコードの例がありますので、それを変更して使用することもできます。

メニューファイル
　USERCMD.MEN　　：値の設定を行うためのコマンドを定義する。
　USEROSM.MEN　　：会話形式で値の設定を行うためのメニューを定義する。

ソースコード
　usrcom.cpp　　：以下の関数を実装する C++ ソースコード。
　acadupi.h　　：以下の関数の宣言を含むインクルードファイル。

　comusrint()　　：定数を初期化する関数。
　comusrset()　　：会話形式で値の表示／変更をする関数。
　comusrsize(), comusrwrite(), comusrread()
　　：定数をモデルファイルに書き込む、またはファイルから読み込む関数。

これらの関数が呼ばれるのは以下の場合です。

comusrint() 関数
　(1)　Advance CAD の起動時
　(2)　モデルの新規開始コマンド
　(3)　モデルの呼出しコマンド（新規モード）でモデルの読み込みの前

comusrset() 関数
　ユーザ定義定数の表示／変更を行うユーザコマンドハンドラ

comusrsize(), comusrwrite(), comusrread() 関数
　(1)　モデルファイルからの読み込み。（モデル呼出しコマンド―新規モード）

(2)　モデルファイルへの書込み。(モデルの保存コマンドおよびサブモデルの作成コマンド)

ユーザ定義定数の値を会話形式で表示／変更するにはユーザコマンドとメニューを追加します。もし、会話形式での変更が不要であれば、コマンドやメニューは必要ありません。ここでは ユーザコマンドを登録する USERCMD.MEN と メニューを追加する USEROSM.MEN について説明します。以下の説明では、2 つの整数、2 つの単精度実数、ひとつの倍精度実数、ひとつの文字列の合計 6 つのユーザ定義定数を定義します。

17.1.1 ユーザコマンドを登録

USERCMD.MEN の例

```
/ AdvanceCAD ver 20 User defined command list
/
/ + [dispatcher#, driver#, form#] !command_name!
/ V [dispatcher#, driver#, form#] !command_name!
/       command name :== [A-Z][A-Z0-9/_]*
/
/ [ 48, n, n] are reserved for User defined modifier
/ [ 32, n, n] are reserved for User defined command
/ [ 53, 98, n] are reserved for User defined command RVP/USER Command.
V [ 53, 98,   0] !RVP/USER!
V [ 53, 98,   1] !USER/INT1!
V [ 53, 98,   2] !USER/INT2!
V [ 53, 98,   3] !USER/TXT!
V [ 53, 98,   4] !USER/RAL1!
V [ 53, 98,   5] !USER/RAL2!
V [ 53, 98,   6] !USER/DBL!
/ End of file
```

最初のコマンド RVP/USER は会話形式でユーザ定義定数の表示／変更を行うためのメニューを呼出すコマンドです。このコマンドを実行するとスクリーンにユーザ定義定数の一覧を表示します。コマンド名は任意ですが、コマンド番号は 53, 98, 0 としなければなりません。

続いてコマンド USER/INT1 から USER/DBL は、個々のユーザ定義定数の値を変更するコマンドです。コマンド名は任意です。コマンドのディスパッチャ番号は 53, ドライバ番号は 98 としなければなりません。フォーム番号は 1 以上の整数です。フォーム番号は ユーザ定義定数の識別に使用します。詳細は comusrset () 関数のところで行います。

17.1.2 メニューを追加

USEROSM.MEN の例

```
/
/ Advance CAD ver 20 User Onscreen menu definition
/ Menu [menu#,category#,screen#,colour#]
/ + <row#,col#> "text" !command! [menu1#,menu2#,colour#]
/ L <row#,col#> "text" !letter!  [menu1#,menu2#,colour#]
/ T <row#,col#> "text" !text!    [menu1#,menu2#,colour#]
/ N <row#,col#> "text" [menu1#,menu2#,colour#,type#,value]
/       type 0=scalar, 1=Mark number, 2=colour number,
/            3=key number, 4=Line font number, 5=Line weight number
/
/ Pagename = rvp_user
Menu    [rvp_user,  3, 10,  c0]
+ < 1, 1> " 例題 "                        !RVP/USER!      [rvp_user, none, c0]
+ < 2, 1> " 整数型＃１          "         !USER/INT1!
+ < 3, 1> " 整数型＃２          "         !USER/INT2!
```

```
+ < 4, 1> ″文字型（１－４文字）″      !USER/TXT!
+ < 5, 1> ″実数型＃１          ″      !USER/RAL1!
+ < 6, 1> ″実数型＃２          ″      !USER/RAL2!
+ < 7, 1> ″倍精度実数型        ″      !USER/DBL!
+ <10, 1> ″終了″                      !RVP/END!
/
/ End of file

Menu [rvp_user,  3, 10,  c0]
```

メニューページ名、カテゴリ番号、表示領域、表示色の定義をします。メニューページ名は任意で、ここでは rvp_user です。カテゴリは 3、表示領域は 10 とします。メニューの表示色は無視されます。

```
+ < 1, 1> ″例題″  !RVP/USER!  [rvp_user, none, c0]
```

ユーザ定義定数の一覧を表示するためのメニューを呼出すコマンドを割付けます。このコマンドが選択されたときに表示するメニューページ名が必要です。この例ではメニューページ名は rvp_user です。メニューページ名が無いとメニューは表示しません。

```
+ < 2, 1> ″整数型＃１          ″    !USER/INT1!
```

個々のユーザ定義定数の値を変更するコマンドをメニューページに割付けます。

```
+ <10, 1> ″終了″                     !RVP/END!
```

会話形式の値変更を終了させるコマンド（RVP/END）を割付けます。RVP/END コマンドは標準のコマンド定義があるので、ユーザが新たに登録する必要はありません。これが選択されると会話形式の値変更を終わり、画面は以前の状態に戻ります。

17.1.3 データ領域の定義

ユーザ定義定数の値を保持する領域を確保します。
この例では、まず、ユーザ定義定数を集めてひとつの構造体として型宣言しています（COMUSR）。構造体でなくてもかまいませんが、ひとまとめにしたほうがわかりやすくなります。構造体のメンバー変数は、double, float, short, char の順に並べると、メンバー変数間にパディングが入らず安全です。
型宣言の次の行がユーザ定義定数を保持する変数 comusr を定義しています。

```
static COMUSR comusr;
```

ファイルスコープの静的変数として定義しています。
データ領域を決める場合は、将来の変更・拡張を考慮しておくことを勧めます。たとえばユーザデータ領域の先頭に、データのバージョン番号などをつけておけば、将来データ領域を変更するときに整合させることができます。

```
// User defined data.
typedef struct {
  double dbl;           // Double precision data
  float  rarr[2];       // Real data array
  short  iarr[2];       // Integer data array
  char   txt[4];        // Text
} COMUSR;

static COMUSR comusr;   // User common area.
```

17.1.4 comusrint () 関数

ユーザ定義定数のデータを初期化する関数です。この例では、構造体 comusr のメンバー変数に値を設定します。

```
void comusrint() {
  comusr.dbl = 100.0;
  comusr.rarr[0] = 10.0f;
```

```
  comusr.rarr[1] = 20.0f;
  comusr.iarr[0] = 1;
  comusr.iarr[1] = 2;
  memcpy(comusr.txt, "ABCD", 4L);
} /* comusrint */
```

17.1.5 comusrset() 関数

定数値の表示／変更をする関数です。下記の例題プログラムを参照しながら説明します。
まず引数の説明をしましょう。

```
void comusrset(int keyset, const int cid[], const TOKEN* token);
```

第 1 引数 keyset は新しい値を設定しかつ表示する (=1) か現在の値の表示だけ (=0) かを指示しています。
第 2 引数 cid は USERCMD.MEN で登録したコマンドのどれかのコマンド番号です。cid[0] は 53、cid[1] は 98 でいつも同じですから考慮しなくてかまいません。cid[2] がフォーム番号で 1 以上の整数です。
フォーム番号 0 のコマンド RVP/USER は除かれることに注意してください。
このフォーム番号から構造体 comusr のメンバー変数と対応付けることができます。

```
form = 1   USER/INT1  comusr.iarr[0]
form = 2   USER/INT2  comusr.iarr[1]
form = 3   USER/TXT   comusr.txt
form = 4   USER/RAL1  comusr.rarr[0]
form = 5   USER/RAL2  comusr.rarr[1]
form = 6   USER/DBL   comusr.dbl
```

第 3 引数 token は新しい値を持つ TOKEN 構造体です。これは第 1 引数 keyset の値が 1 の時だけ使用します。そうでないときは参照してはいけません。

最初にユーザが RVP/USER コマンドを実行したとき、メニューページの指定があれば、スクリーンにユーザ定義定数の一覧を表示します。このとき、メニューページ内の項目に対し現在値を表示するためこの関数が呼ばれます。この場合は値の表示だけなので keyset の値は 0 です。項目ひとつに対して 1 回の呼び出しが行われます。このメニューページ例では 6 回です。次に、ユーザが一覧のなかのメニュー項目を選択し新しい値を入力すると、ユーザ定義定数の値を更新し表示するためこの関数が呼ばれます。keyset の値は 1 です。実際に値を表示するには、後述する Rvpdisp() 関数を呼び出すことで行います。この関数はコマンド番号に対応付けたられた構造体 comusr のメンバー変数の値の更新／取得を行うことです。

```
/*
 * Set and/or Display parameters for the User defined data.
 * Command syntax.
 *   RVP/USER [command value]* RVP/END
 * Input Arguments
 *   keyset  Setting control switch.
 *            0 : Display only
 *            1 : Set and Display.
 *   cid     Command number. cid = { 53, 98, form }
 *            form = 1 USER/INT1
 *            form = 2 USER/INT2
 *            form = 3 USER/TXT
 *            form = 4 USER/RAL1
 *            form = 5 USER/RAL2
 *            form = 6 USER/DBL
 *   token    Token. (This is NULL if keyset is 0)
 *  Return Error code. 0 : No error
 */
int comusrset(int keyset, const int cid[], const TOKEN* token) {
  #define RVPCTG 3
```

```
if (keyset == 1) {
  // Set value to the defined data.
  switch (cid[2]) {
  case 1: // Integer data #1.
    if (token->typ != TknSCL)
      return 1;
    if (fabs(token->scl) > 32767.0)
      return 1;
    comusr.iarr[0] = short(token->scl);
    break;
  case 2: // Integer data #2.
    if (token->typ != TknSCL)
      return 1;
    if (fabs(token->scl) > 32767.0)
      return 1;
    comusr.iarr[1] = short(token->scl);
    break;
  case 3: // Text data #1.
    if (token->typ != TknTXT)
      return 1;
    if (token->txt[0] == '¥0' || strlen(token->txt) > sizeof(comusr.txt))
      return 1;
    memset(comusr.txt, ' ', sizeof(comusr.txt));
    memcpy(comusr.txt, token->txt, strlen(token->txt));
    break;
  case 4: // Float data #1.
    if (token->typ != TknSCL)
      return 1;
    comusr.rarr[0] = float(token->scl);
    break;
  case 5: // Float data #2.
    if (token->typ != TknSCL)
      return 1;
    comusr.rarr[1] = float(token->scl);
    break;
  case 6: // Double precision data #1.
    if (token->typ != TknSCL)
      return 1;
    comusr.dbl = token->scl;
    break;
  default:
    return 1;
  }
}

// Get value.
int mode, im3swt = 0;
char itext[4];
double v;
switch (cid[2]) {
  case 1: // Integer data #1.
    v = (double)comusr.iarr[0];
    mode = TknSCL;
    break;
  case 2: // Integer data #2.
    v = (double)comusr.iarr[1];
    mode = TknSCL;
    break;
  case 3: // Text data #.
    memcpy(itext, comusr.txt, sizeof(itext));
    mode = 10 * sizeof(comusr.txt);
    break;
```

```
      case 4: // Float data #1.
        v = comusr.rarr[0];
        mode = TknSCL;
        im3swt = 1;
        break;
      case 5: // Float data #2.
        v = (double)comusr.rarr[1];
        mode = TknSCL;
        im3swt = 1;
        break;
      case 6: // Double precision data #1.
        v = comusr.dbl;
        mode = TknSCL;
        im3swt = 1;
        break;
      default:
        return;
    }

    // Display a parameter.
    if (cid[2] == 3) {
      Rvpdisp(RVPCTG, cid, mode, itext, keyset, im3swt);
    } else {
      Rvpdisp(RVPCTG, cid, mode, &v, keyset, im3swt);
    }

    return 0
  } // comusrset
```

17.1.6 Rvpdisp() 関数

ユーザ定義の定数を表示します。comusrset() 関数以外では使用できません。

```
void Rvpdisp(int ictg, const int cid[], int mode, const void* vdata, int ieras, int im3swt);
```

ictg	コマンドのカテゴリ番号。常に 3 にする。
cid	コマンド番号（comusrset() 関数の引数をそのまま渡す）
mode	表示するデータの種類。 3 : 数値 文字数 *10 : 文字列
vdata	表示するデータ (引数 mode による） 倍精度実数 (double *) または 文字列 (char *)
ieras	表示する前に表示部分を消すかどうか。 1 : 消す 0 : 消さない
im3swt	数値 (mode=3) の時に小数点を表示するかどうか。 1 : 表示する 0 : 表示しない

17.1.7 ユーザ定義定数のモデルファイルへの保存

ユーザ定義定数をモデルファイル に書き込むことができます。またモデルファイルを読み込むときには、モデルファイルに保存した値を復元します。

モデルファイル保存は、comusrsize() 関数を呼出し、ファイルに書き込むユーザ定義定数のデータサイズを問い合わせます。comusrsize() 関数がサイズ 0 を返せば保存は行いません。そうでなければ、保存のため comusrwrite() 関数を呼び出します。

モデルファイル読込みは、モデルファイルにユーザ定義定数のレコードが現れたら、comusrsize() 関数を呼出しユーザ定義定数のデータサイズを問い合わせます。comusrsize() 関数が返したデータサイズがユーザ定義定数レコードのサイズと等しい時だけ comusrread() 関数を呼び出します。それ以外では保存した値の復元は行いません。

17.1.8 comusrsize() 関数

ユーザ定義定数のデータサイズを返す関数です。ユーザ定義定数を定義してもモデルに保存しない場合は、comusrsize() 関数がサイズ 0 を返すようにします。

下記の例題プログラムでは、モデルファイルへの入出力をしないよう 0 を返すようにしています。

```
short comusrsize() {
  return 0; // Don't read/write the user defined data.
}
```

この行を変更して正しいデータサイズを返えすようにすれば、モデルファイルへの入出力を行います。データサイズは short データの単位（2 バイトを 1 とする）で数えます。データサイズは 1 から 32767 までです。データは文字列、整数、実数、倍精度実数の種類ごとに分けて処理しなければなりません。文字列の長さは偶数でなければならないことに注意してください。

構造体 comusr の場合は次のようになります。

```
(sizeof(comusr.dbl) + sizeof(comusr.rarr) + sizeof(comusr.iarr) + sizeof(comusr.txt)) / 2;
```

構造体のサイズを使って sizeof(comusr) / 2 としたくなりますが、構造体のメンバー変数間にパディングが入ると実際のサイズと異なってしまいます。

17.1.9 comusrwrite() 関数

この関数はモデルファイルにユーザ定義定数を書き込みます。データは文字列、整数、実数、倍精度実数の種類ごとに分けて出力しなければなりません。出力データサイズの総和は comusrsize() 関数が返す値と同じでなければなりません。そうでないとユーザ定義定数レコードのヘッダーの持つレコードサイズとデータ部のサイズが異なることになり、モデルファイルの読み込みが失敗します。
comusrwrite() 関数の引数は AwUserIO オブジェクトのポインタです。このオブジェクトの write メソッドを使ってモデルファイルへ書き込みます。
下記に例を示します。

```
int comusrwrite(AwUserIO* writer) {
  int ier = 0;
  if ((ier = writer->WriteDouble(1, &comusr.dbl)) != 0)// Write one double.
    return ier;
  if ((ier = writer->WriteFloat(2, comusr.rarr)) != 0)// Write two float.
    return ier;
  if ((ier = writer->WriteShort(2, comusr.iarr)) != 0)// Write two short.
    return ier;
  if ((ier = writer->WriteChar(4, comusr.txt)) != 0)// Write four char.
    return ier;
  return ier;
}
```

17.1.10 comusrread() 関数

この関数はモデルファイルからユーザ定義定数を読み込みます。comusrwrite() 関数で書き込んだのと同じ順序で文字列、整数、実数、倍精度実数の種類ごとに分けて入力しなければなりません。入力データサイズの総和は comusrsize() 関数が返す値と同じでなければなりません。そうでないとユーザ定義定数

レコードのヘッダーの持つレコードサイズとデータ部のサイズが異なることになり、モデルファイルの読み込みが失敗します。
comusrread() 関数の引数は AwUserIO オブジェクトのポインタです。このオブジェクトの read メソッドを使ってモデルファイルから読み込みます。
下記に例を示します。

```
int comusrread(AwUserIO* reader) {
  int ier = 0;
  if ((ier = reader->ReadDouble(1, &comusr.dbl)) != 0)// Read one double.
    return ier;
  if ((ier = reader->ReadFloat(2, comusr.rarr)) != 0)// Read two float.
    return ier;
  if ((ier = reader->ReadShort(2, comusr.iarr)) != 0)// Read two short.
    return ier;
  if ((ier = reader->ReadChar(4, comusr.txt)) != 0)// Read four char.
    return ier;
  return ier;
}
```

17.1.11 AwUserIO クラス

このオブジェクトはモデルファイルにデータを書き込むメソッドとモデルファイルからデータを読み込むメソッドを持ちます。

書き込みメソッド

```
int WriteDouble(int count, const double dpfp[]);
int WriteFloat(int count, const float spfp[]);
int WriteInt(int count, const int iarr[]);
int WriteShort(int count, const short sarr[]);
int WriteChar(int count, const char carr[]);
```

読み込みメソッド

```
int ReadDouble(int count, double dpfp[]);
int ReadFloat(int count, float spfp[]);
int ReadInt(int count, int iarr[]);
int ReadShort(int count, short sarr[]);
int ReadChar(int count, char carr[]);
```

データタイプごとのメソッドがあります。これらのメソッド内でバイトオーダー処理を行いますのでデータタイプごとに適切なメソッドを使用しなければなりません。(モデルファイルは big endian でなければなりません)
全てのメソッドの第 1 引数はデータ数 (1-)、第 2 引数がデータへのポインタです。文字列では必ず偶数バイト ((count+1)/2) 処理します。WriteChar メソッドは count が 1 または 2 のときは 2 バイト、3 または 4 の時は 4 バイト出力します。
メソッドは成功したとき 0 を返します。

17.2 モデル管理のカスタマイズ

ソースコード usrmdm.cpp には、以下の関数が含まれています。これらの関数はこのままでは何も処理しない空の関数ですが、関数を書き換えることによりユーザ独自の処理を追加できます。

ソースコード

usrmdm.cpp	：以下の関数を実装する C++ ソースコード。
acadupi.h	：以下の関数の宣言を含むインクルードファイル。
mdm001()	:MDM/TOOL コマンドが呼び出す関数。ユーザが作成したファイル管理プログラムにアクセスするためのもの。

mdm101() ：モデルファイル及びサブモデルファイルに関する操作が行われた時に行う処理を実装するコールバック関数。

これらの関数が呼ばれるのは以下の場合です。
mdm001() 関数
(1) MDM/TOOL コマンド
mdm101() 関数
(1) モデルファイルの保存、呼出し
(2) サブモデルの保存、配置
(3) ファイルの管理

17.2.1 mdm001() 関数

この関数は割り込みコマンド MDM/TOOL が選択されたときに呼び出されます。ユーザは独自のファイル管理プログラムを起動するなどの処理を実装することができます。

デフォルトの実装は下記の通りで、何もしません。

```
void mdm001(char* ctext, size_t len_ctext) {
  *ctext = '¥0';
  return;
} // mdm001
```

第 1 引数は出力引数 ctext、第 2 引数は文字配列 ctext のサイズです。char ctext[len_ctext] の意味になります。ctext は null-char で終了させます。上記の例では空文字列を返します。ここで有効な文字列を設定した場合、その文字列が次のコマンド入力となります。MDM/TOOL コマンドにより中断された優先度の低いコマンドにテキスト入力を与えたい場合に使います。

17.2.2 mdm101() 関数

この関数はモデルファイルおよびサブモデルファイルに関する操作が行われたときに呼び出されます。通常のファイル管理にユーザ独自の処理を追加することができます。

デフォルトの実装は下記の通りで、何もしません。

```
void mdm101(int otyp, const char* fname, const char* oname) {
  return;
} /* mdm101 */
```

第 1 引数 otyp は操作を表す整数です。

100 : アクティブモデルの保存
101 : アクティブモデルの部分的な保存。またはサブモデルファイルの作成
102 : ピクチャ書き込み
103 : 所有者名の変更
104 : 削除
105 : ファイル名の変更
106 : コピー
107 : テープからのコピー
200 : モデルファイルの呼出し
201 : 部分的な呼出し。またはサブモデルの配置や更新による再配置

第 2 引数はモデルファイル名。第 3 引数は操作に依存し、以下にあげる場合だけ有効です。

所有者名の変更 (otyp == 103) : 新しい所有者名
ファイル名の変更 (otyp == 105) : 元のモデルファイル名
コピー (otyp == 106) : 元のモデルファイル名

第 18 章　ユーティリティ

18.1 AwEucEncoder クラス

文字コードは日本語 EUC（EUC-JP）です。使用できる文字は下記の通りです。

⑴　**改行文字**

ASCII コード (0x0D、'\r') が行の区切り。

⑵　**ASCII 文字**

バイト　0x20 ～ 0x7E　　(#0xxxxxxx)

空白および "!" から " ～ " までの印字可能文字のみを使用します。

⑶　**日本語文字** (JIS X0208)

1 バイト目 (B1) 0xA1 ～ 0xFE　(#1xxxxxxx)

2 バイト目 (B2) 0xA1 ～ 0xFE　(#1xxxxxxx)

JIS 区点番号との対応は次の通りです。

JIS 区 = B1 - 160 = (1 ～ 94)、JIS 点 = B2 - 160 = (1 ～ 94)

⑷　**日本語文字**（JIS X0201）

1 バイト目 (B1) 0x8E　　(#10001111)

2 バイト目 (B2) 0xA1 ～ 0xFE　(#1xxxxxxx)

1 バイト目は、JIS X0201 を示すシングルシフトです。

俗にいう半角カタカナです。SXF 互換のため追加しましたが、それ以外での使用はお勧めしません。

⑸　**メタ文字**

メタ文字は ASCII テキストフォントに製図用文字を追加すること、特殊な表記を制御する文字として使用します。

1 バイト目 (B1) 0x8D　　(#10001101)

2 バイト目 (B2) 0xA1 ～ 0xFE　(#1xxxxxxx)

1 バイト目は、つぎの 1 バイトがメタ文字であることを示すシングルシフトです。2 バイト目の第 1 ビットを除いた下 7 ビットは ASCII 文字の "!" から " ～ " に対応します。ASCII は 7 ビットですがこれを 8 ビットまで拡張し、拡張した部分をメタ文字と呼びます。通常の ASCII 文字と区別するため、"\M" をつけて \MA, \MB などと記述します。\MA は ASCII 文字 "A" に対応するメタ文字です。

メタ文字の用途のひとつは ASCII テキストフォントに以下に示す製図用文字を追加することです。追加した文字を表現するのにメタ文字を使っています。

度記号	\M0
径記号	\M1
プラスマイナス	\M2
角記号	\M3
弧記号	\M4

もうひとつの用途は、文字列の特殊な表記を制御する文字として使います。

寸法値	\MD 文字列 \MZ
分数	\MF 分子 \MY 分母 \MZ
２段文字	\MS 上段文字列 \MY 下段文字列 \MZ
	\MS 上段文字列 \MZ

\MS \MY 下段文字列 \MZ
区切り文字として、\MY（左寄せ）以外に \MX（中央寄せ）、\MW（右寄せ）があります。
マーク挿入　　\MM マーク番号, マーク番号,…, マーク番号 \MZ

メタ文字を示すシングルシフト（0x8D）は日本語 EUC の規則を守っていませんので、グラフィックステキスト以外で使用するのは避けたほうがよいでしょう。将来はこのシングルシフト（0x8D）を止め "\M" 表記に変えるなどすべきでしょう。

18.1.1 文字コード変換

MS 漢字コード（Shift_JIS) を日本語 EUC（EUC-JP）に変換、その逆変換をします。

Shift_JIS を EUC-JP に変換します。
```
static std::string Sjis2Euc(const std::string& src, size_t maxlen = std::string::npos);
```
src　　Shift_JIS マルチバイト文字列。
maxlen　変換後の文字列の最長バイト数。省略時は制限しません。
戻り値　EUC-JP マルチバイト文字列。

EUC-JP を Shift_JIS に変換します。
```
static std::string Euc2Sjis(const std::string& src, size_t maxlen = std::string::npos);
```
src　　EUC-JP マルチバイト文字列。
maxlen　変換後の文字列の最長バイト数。省略時は制限しません。
戻り値　Shift_JIS マルチバイト文字列。

18.1.2 イテレータ

日本語を含む文字列から順番に 1 文字を取出していきます。
```
static int NextChar(const std::string& src, size_t index, int* type, unsigned char code[]);
```
src　　文字列。
index　評価開始位置のインデックス (0 <= index < src.lengt)。
type　　評価した文字の種類を示す整数。
code　　評価した文字のコードを格納する長さ 4 の配列。
戻り値　評価したバイト数。

定数名	意味	code	nbyte
TYPE_UNKNOWN	終了か不正な文字	{0x0, 0x0, 0x0, 0x0}	終了は 0、不正は 1-4
TYPE_NEWLINE	改行	{0x0, 0x0, 0x0, 0x0D}	1
TYPE_CONTROL	ASCII 非印字文字	{0x0, 0x0, 0x0, column} coulmn : ASCII code (0-9,11-12,14-31,127)	1
TYPE_PRINTABLE	ASCII 印字文字	{0x0, 0x0, 0x0, column} 0　　: 空白文字 1-94 : 印字文字 95-99: 製図記号（拡張）	ASCII は 1、製図記号は 2

定数名	意味	code	nbyte
TYPE_FUNCTION	メタ文字 (A - Z)	{0x0, 0x0, 0x0, column} column 1-26	2
TYPE_MULTIBYTE	EUC-JP codeset1, 3	{0x0, plane, row, column} plane 1-16 row 1-94 column 1-94	2-4
TYPE_HANKANA	EUC-JP codeset2	{0x0, 0x1, 0x0, column} column 1-94	2

例）文字列 src のメタ文字を ”\Mx” 形式の文字列に変えます。ASCII 非印字文字や不正な文字があれば取り除きます。

```
std::string dst;
int nbyte = 0;
for (size_t j = 0; j < src.length(); j += nbyte) {
  int type;
  unsigned char code[4];
  nbyte = AwEucEncoder::NextChar(src, j, &type, code);
  if (nbyte <= 0)
    break;
  if (type == AwEucEncoder::TYPE_UNKNOWN) {
    // Remove illegal char.
  } else if (type == AwEucEncoder::TYPE_CONTROL) {
    // Remove non-printable char.
  } else if (type == AwEucEncoder::TYPE_NEWLINE) {
    dst.append("¥n");
  } else if (type == AwEucEncoder::TYPE_PRINTABLE && 95 <= code[3]) {
    dst.append("¥¥M");
    dst.push_back(code[3] - 95 + '0');
  } else if (type == AwEucEncoder::TYPE_FUNCTION) {
    dst.append("¥¥M");
    dst.push_back(code[3] + '@');
  } else {
    dst.append(src, j, nbyte);
  }
}
```

18.2 AwStringUtil クラス

文字列への変換。printf() 関数相当。

```
static std::string Format(const char* format, ...);
```

format　　変換制御文字列。
可変引数　format と一致する引数。
戻り値　　結果の文字列。

水平タブを展開します。文字列中の全ての水平タブを適切な数の空白文字でおきかえます。

```
static std::string Detab(const std::string& src, int tabsize = 8);
```

src　　入力文字列。
tabsize　水平タブの幅。省略時は８カラム。
戻り値　変換後の文字列。

カンマ区切りの数値が並んだ文字列を数値に変換します。

```
static int ParseDouble(const std::string& src, int maxcnt, double dpfp[]);
```
src　　入力文字列。
maxcnt　配列のサイズ。
dpfp　　変換した数値を格納する配列 dpfp[maxcnt]。
戻り値　変換した数値の個数 (0 - maxcnt)。

入力文字列の評価規則
- **数値の区切りはカンマ。**
- **数値の区切りの間に何も文字がないときや空白だけのときは数値 0 とする。また入力文字列の最初にカンマが表われたときや最後にカンマが表われた時も同様。**
- **数値に変換できない文字列は数値 0 とする。**
- **一つの数値の文字列長さは 20 文字以内。**

例
```
"1, 2.0 ,3 "    ->    1.0, 2.0, 3.0
", 2, ,4,"      ->    0.0, 2.0, 0.0, 4.0, 0.0
"1e2,,3d-1"     ->    100.0, 0.0, 0.3
"1,abc,3,xyz4"  ->    1.0, 0.0, 3.0, 0.0
" , "           ->    0.0, 0.0
```

文字列の空白文字を見つけ、その空白文字を含め以降を削除します。
```
static void TruncateAtSpace(std::string* string);
```
string　入力文字列。この文字列を直接変更します。空白文字がなければ変更しません。

18.3 AwFile クラス

ファイルパスは以下のように表現します。
ディレクトリ区切り文字はスラッシュ (/)。
文字コードは EUC-JP。

Windows のファイルパスはプログラム上ではバックスラッシュをスラッシュに置き換えたものにします。
```
C:/acad/files/SAMPLE.MDL
```
このようなプラットホームに依存しないファイルパスを抽象パスと呼びます。本書では特にことわらない限りファイルパスは抽象パスです。

ここではファイルパスに関して下記の用語を使用します。
- **ファイルパス （ディレクトリパス + ファイル名）**
- **ディレクトリパス**
- **ファイル名 （名前 + 拡張子）**
- **名前**
- **拡張子（ドットから始まる）**

ファイルパス "C:/acad/files/SAMPLE.MDL" は、ディレクトリパス "C:/acad/files/"、ファイル名 "SAMPLE.MDL"、名前 "SAMPLE"、拡張子 ".MDL" となります。
トップディレクトリから始まるファイルパスをフルパスと呼びます。”C:” で始まるファイルパスのことです。
複数のドット (‘.’) を含む名前は混乱します。複数のドットがある場合は最後のドットを拡張子の開始文字とします。"C:/acad/files/A.B.C.MDL" は 名前が "A.B.C" 、拡張子は ".MDL" と解釈します。

このクラスはファイル名処理とファイル操作を行う ユーティリティクラスです。

18.3.1 抽象パスを調べる

ディレクトリパスを得ます。
```
static std::string GetDirectory(const std::string& path);
```
戻り値　ディレクトリパスを返します。ディレクトリパスがなければ空文字列です。

ファイル名を得ます。
```
static std::string GetFilename(const std::string& path);
```
戻り値　ファイル名を返します。ファイル名がなければ空文字列です。

ファイルの名前を得ます。
```
static std::string GetName(const std::string& path);
```
戻り値　ファイルの名前を返します。名前がなければ空文字列です。

ファイルの拡張子を得ます。
```
static std::string GetExtension(const std::string& path);
```
戻り値　ファイルの拡張子を返します。拡張子がなければ空文字列です。

フルパスか判定します。
```
static bool IsFullPath(const std::string& path);
```
戻り値　フルパスであれば true を返します。

ディレクトリパスの長さを得ます。
```
static int GetDirectoryLength(const std::string& path);
```
戻り値　ディレクトリパスのバイト数を返します。ディレクトリパスが無ければ 0 です。

ファイル名の位置を得ます。
```
static int FindName(const std::string& path);
```
戻り値　ファイル名の開始位置のインデックスを返します。0 <= index < path.length()。ファイル名が無ければ -1 です。

ファイルの拡張子の位置を得ます。拡張子はドット ('.') で始まります。
```
static int FindExtension(const std::string& path);
```
戻り値　ファイル拡張子の開始位置のインデックスを返します。0 <= index < path.length()。拡張子が無ければ -1 です。

18.3.2 ファイルパス操作

ファイル名補完はファイル名の不足する部分を補完してフルパスにすることです。ディレクトリパスやファイル拡張子が無ければそれらが付いたものにします。このとき使用するのがファイルタイプです。ファイルタイプの詳細は『システム管理者の手引き』「第２章起動と環境設定」の「コンフィグレーションファイル」を参照してください。
ファイルタイプは以下のように、タイプごとのディレクトリパス、ファイル拡張子を指定するものです。
```
#SYMBOL#        "C:/acad/files/"        !.SYM!
#MENU:INP#      "C:/acad/menu/MENU"     !.INP!
```

ファイルタイプ名は #SYMBOL# や #MENU:INP# のようにシャープ（'#'）で囲まれた文字列です。ディレクトリパス "C:/acad/menu/MENU" はデフォルトのファイルの名前 MENU が付いています。

ファイル名を補完しフルパスにします。
```
static std::string ToFullPath(const std::string& type, const std::string& filename,
                              bool ignorename = false, bool allownocateg = false);
```
type　ファイルタイプ名。このファイルタイプのディレクトリパスやファイル拡張子を補完に用います。

filename　補完する抽象パス。
ignorename　ファイル名 (filename) に名前が含まれないときの処理を指示します。デフォルトのファイルの名前が有ってもそれを使用しないなら true とします。
allownocateg　ファイルタイプ (type) が登録してないときの処理を指示します。タイプがなくてもかまわないなら true とします。そうでなければタイプがないのでエラーです。
戻り値　抽象フルパス。失敗したときは空文字列です。

下記の間接パスは実際のディレクトリパスに展開します。
~/　ホームディレクトリ
./　カレントディレクトリ
../　親ディレクトリ
~xxx/　ユーザ xxx のホームディレクトリ
Windows では下記のパスも扱えます。
drive:/dir　Drive name
//node/dir　Universal name

#SYMBOL# での補完例。

入力	結果
"ABC"	"C:/acad/files/ABC.SYM"
"ABC.TST"	"C:/acad/files/ABC.TST"
"/tmp/ABC"	"/tmp/ABC.SYM"
"/tmp/ABC.TST"	"/tmp/ABC.TST"

ファイルパスの拡張子を変更します。
```
static void ReplaceExtension(std::string* path, const std::string& ext);
```
path　入力抽象パス。このパスを直接変更します。
ext　新しい拡張子（ドットで始まること)。拡張子を除去したければ空文字列を与えます。

プラットホーム依存パスを抽象パスにします。
```
static std::string ToAbstractPath(const std::string& path);
```
path　プラットホーム依存パス。
戻り値　抽象パス。

抽象パスをプラットホーム依存パスにします。
```
static std::string ToPlatformPath(int codeset, const std::string& path);
```
codeset　依存パスの文字コード（0= 無変換、1=EUC-JP、2=Shift_JIS)。
path　抽象パス。
戻り値　プラットホーム依存パス。

18.3.3 ファイル操作

ファイルの accessibility を調べます。access() 関数相当。
```
static int Access(const std::string& path, int amode);
```
path　対象の抽象パス。
amode　accessibility。F_OK、R_OK、W_OK、X_OK。
戻り値　許可するなら 0 を返します。

ファイルを削除します。unlink() 関数相当。
```
static int Remove(const std::string& path);
```
path　削除するファイルの抽象パス。
戻り値　削除したなら 0 を返します。

ファイル名を変更します。rename() 関数相当。
```
static int Rename(const std::string& oldname, const std::string& newname);
```
oldname　現在の抽象パス。

newname　新しい抽象パス。
戻り値　成功したなら 0 を返します。

ファイルの Status を得ます。stat() 関数相当。
```
static int Stat(const std::string& path, struct stat* status);
```
path　対象の抽象パス。
status　stat 構造体へのポインタ。
戻り値　成功したなら 0 を返します。

ディレクトリかどうか調べます。
```
static bool IsDirectory(const std::string& path);
```
path　対象の抽象パス。
戻り値　ディレクトリなら true を返します。

ファイルをバックアップファイルにリネームします。バックアップファイルのパスは元のファイルパスの最後に ".OLD" を付けたものです。
```
static int MakeBackup(const std::string& type, const std::string& path);
```
type　ファイルタイプ名または空文字列。
path　リネームするファイルの抽象パス。
戻り値　リネームしたなら 0 を返します。

18.4 AwFileIO クラス

このクラスはファイル入出力操作を行います。ファイル入出力は C 言語標準ライブラリのファイル入出力関数を使って出来ますが、主に下記の理由でこのクラスを使います。
(1)　ファイルパスは抽象パスをそのまま渡せる（AwFile クラス参照）。
(2)　デストラクタがファイルのクローズを確実にする。
(3)　テキストファイルの入出力で日本語 文字コード変換を自動的に行う。

```
AwFileIO();
```
コンストラクタ。後述の日本語 文字コード変換は有効 (true) です。

```
~AwFileIO();
```
デストラクタ。ファイルが開いた状態であれば、それをクローズします。

18.4.1 テキストファイル

C 言語標準ライブラリ関数の fopen()、fclose()、fgets()、fputs()、fprintf() 関数に相当するメソッドがあります。

プログラムで扱う文字コードは日本語 EUC（EUC-JP）に統一しているので、Shift_JIS コードのテキストファイルを扱うときには日本語 文字コード変換（Shift_JIS と EUC-JP の変換）が必要になります。これを簡単にするために、日本語 文字コード変換を Fgets()、Fputs()、Fprintf() メソッドに行わせることができます。
Fgets()
ファイルから読み込み込んだ文字列（Shift_JIS）を EUC-JP に変換します。
Fputs()、Fprintf()
文字列（EUC-JP）を Shift_JIS に変換してファイルに書き込みます。

Fgets()、Fprintf() に 文字コード変換を行わせる場合には一度に処理できる文字数はおよそ 500 バイト以下の制限があります。ASCII 文字だけを出力する場合など、日本語 文字コード変換をさせたくないときは SetCodeConversion(false) を呼出し文字コード変換を無効にできます。

日本語 文字コード変換を制御します。

```
void SetCodeConversion(bool enable);
```
enable　日本語 文字コード変換を有効 (true) ／無効 (false) にします。

ファイルをオープンします。fopen() 関数相当。
```
bool Fopen(const std::string& path, const char* mode);
```
path　抽象パス。
mode　アクセスモードを指定する文字列 “r”, “w”, “a” など。fopen() 関数を参照。
戻り値　成功したときは true を返します。

ファイルをクローズします。fclose() 関数相当。
```
void Fclose();
```

ファイルから最大 (n-1) 文字を読み込みます。fgets() 関数相当。
```
char* Fgets(char* s, int n) const;
```
s　char 配列。
n　最大文字数（バイト）。
戻り値　成功したときはポインタ s を返します。それ以外は null ポインタを返します。

ファイルに文字列を書き込みます。fputs() 関数。
```
int Fputs(const char* s) const;
```
s　出力文字列。
戻り値　成功したときは 0 以上の値を返します。それ以外は EOF を返します。

書式変換した文字列をファイルに書き込みます。fprintf() 関数相当。
```
int Fprintf(const char* format, ...) const;
```
format　変換制御文字列。
可変引数　format と一致する引数。
戻り値　出力バイト数を返します。エラーのときは負の値を返します。

18.4.2 バイナリファイル

C 言語標準ライブラリ関数の open()、close()、read()、write() 関数に相当するメソッドがあります。

ファイルをオープンします。open() 関数相当。
```
bool Open(const std::string& path, int oflag, int mode);
```
path　抽象パス。
oflag　アクセスモード。open() 関数を参照。
mode　ファイル作成モード。open() 関数を参照。
戻り値　成功したときは true を返します。

ファイルをクローズします。close() 関数相当。
```
void Close();
```

ファイルから nbyte のデータを読み込みます。read() 関数相当。
```
int Read(void* buf, int nbyte) const;
```
buf　データを格納する配列。
nbyte　読み込むデータのバイト数。
戻り値　成功したときは読み込んだバイト数を、失敗したときは -1 を返します。

ファイルに nbyte のデータを書き込みます。write() 関数相当。
```
int Write(const void* buf, int nbyte) const;
```
buf　書き込むデータ。
nbyte　書き込むデータのバイト数。

戻り値　　成功したときは書き込んだバイト数を、失敗したときは -1 を返します。

18.5 AwUtil クラス

18.5.1 バイト列

指定した位置のバイトを取り出します。
```
static short GetByte(const char* bytes, int nbyte);
```
bytes　バイト列を格納した配列。
nbyte　バイトの位置を指定する整数 (1-)。
戻り値　バイトの値 (0-255)。

指定した位置のバイトの値を設定します。
```
static void  SetByte(char* bytes, int nbyte, int value);
```
bytes　バイト列を格納する配列。
nbyte　バイトの位置を指定する整数 (1-)。
value　バイトの値 (0-255)。

18.5.2 ビット列

指定したビットフィールドを取り出します。
```
static short GetBits(const short* bits, int sbit, int ebit);
```
bits　ビット列を格納した short int 配列。
sbit　開始ビット位置を指定する整数 (1-)。
ebit　終了ビット位置を指定する整数 (1-)。sbit <= ebit。sbit / 16 == ebit / 16。
戻り値　ビットフィールドの値。

指定したビットフィールドの値を設定します。
```
static void  SetBits(short* bits, int sbit, int ebit, int value);
```
bits　ビット列を格納する short int 配列。
sbit　開始ビット位置を指定する整数 (1-)。
ebit　終了ビット位置を指定する整数 (1-)。sbit <= ebit。sbit / 16 == ebit / 16。
value　ビットフィールドの値。下位 N ビット (N = ebit - sbit + 1) を使います。

ビットの位置は 1 から始まる整数です。ビットフィールドを指定する開始ビット位置と終了ビット位置は同じ short int になければなりません。例えば GetBits(bits, 15、17) は bits[0] と bits[1] にまたがるのでエラーで、戻り値は 0 になります。

18.5.3 日付け

現在の日付けを得ます。
```
static std::string GetCurrentDateString();
```
戻り値　日付の文字列。
```
static void GetLocalTime(short date[], short time[]);
```
date　日付を格納する配列。
time　時刻を格納する配列。

日付けを文字列に変換します。
```
static std::string GetDateString(const short date[], const short time[]);
```
date　日付を格納した配列。
time　時刻を格納した配列。
戻り値　日付の文字列。

日付を格納する配列は
date[0] : 日 (1-31)、date[1] : 月 (1-12)、date[2] : 西暦年 (1986 - 2068)
time[0] : 時 (0-23)、time[1] : 分 (0-59)、time[2] : 秒 (0-59)
日付の文字列は ”yyyy/mm/dd hh:mm:ss” の形式で長さ 19 文字です。dd と hh の間に空白文字があります。

18.6 STL Algorithm

C++ 標準テンプレートライブラリ (STL) のアルゴリズムにはすぐに使える便利な機能があります。例えば下記ののようなものです。

2 つの要素のデータを交換する	std::swap()
2 つの配列の内容を交換する	std::sawp_range()
配列にある値を設定する	std::fill()
ソート	std::sort()

STL のアルゴリズムを使用するときはソースコードの先頭に #include <algorithm> を追加します。

18.7 関数

● 関数一覧

macroload	**マクロファイルをコンパイル、ロード、実行する。**
mcrcalc	**計算器**
FilSelect	**ファイル名を画面に表示し、該当するものを選ぶ。**

18.7.1 マクロファイルをコンパイル・ロード・実行する

マクロファイルをコンパイル、ロード、実行する。

【呼出し形式】

```
int macroload(int swt, const char* name)
```

【入力引数】

swt　再ロードするかどうかのスイッチ (0, -2)
　0　: 指定されたマクロが既にロードされていれば再ロードしない。実行のみする。
　-2　: 指定されたマクロ既にロードされているかいないに関わらず再度コンパイル、ロードし実行する。

name　マクロ名。(Null-char terminated)
ディレクトリ名とファイル拡張子は付けない。ACAD.SET の #MACRO# で指定されているディレクトリ名とファイル拡張子を、この関数自身で付加している。

【戻り値】

0　:　正常。
1　:　コンパイルエラー。
2　:　エラー（マクロが存在しない。リスタート中。）

注意
この関数は指定されたマクロソースをコンパイルし、エラーがなければロードし、マクロの実行フラグをオンにする。実行はこの関数を呼び出したユーザプログラムが処理を終了して Advance CAD の入力処理に戻ったときにロードされたマクロを実行する。

マクロのコンパイル結果は Advance CAD の起動ディレクトリの macro.lis に出力される。

18.7.2 計算器

計算器。この関数は Advance CAD の計算器で、キーボードから計算式を '[' と ']' でくくって入力すると呼び出される。
ユーザプログラムでこの関数を使用するのは主に以下の目的である。

- **計算器の変数に値を割り付ける。**
- **計算器の変数の値を参照する。**

計算器の変数はマクロの外部変数となる。マクロの内部変数にはアクセスできない。

【呼出し形式】

```
int mcrcalc(const char* expr, double* dpfp, char* cstr, size_t cstr_len)
```

【入力引数】

expr		計算する式 (Null-char terminated)
	(1)	'[' と ']' は不要
	(2)	計算器の変数名は英大文字と数字のみ
	(3)	数値定数 整数型、実数型いずれでもよいが、すべて実数型に変換してから計算する。
	(4)	文字定数はクォーテーション (") で囲む
cstr_len		配列 cstr の長さ。

【出力引数】

dpfp	計算結果が数値のとき値が設定される。
cstr	計算結果が文字列のときにそれを格納する配列。(Null-char Terminated)

【戻り値】

計算結果のデータの型またはエラーコード

0	:	なし　(たとえば clear() の結果)
3	:	数値　(たとえば a=2*pi の結果)
4	:	文字列　(たとえば a="ABC" の結果)
30	:	配列　(たとえば a=array(3) の結果) dpfp は配列のサイズ。
-201	:	式が誤り
-205	:	式に未定義のレジスタがある
-208	:	式に未定義の変数がある
-212	:	式に未定義の組み込み関数がある

例

```
char cstr[16];
double dpval;

/* 計算器の変数 A に数値 10 を割り付ける。*/
const char* s = "a=10";
mcrcalc(s, &dpval, cstr, sizeof(cstr));

/* 計算器の変数 B に文字列 'ABC' を割り付ける。*/
s = "b=¥"ABC¥"";
mcrcalc(s, &dpval, cstr, sizeof(cstr));

/* 計算器の変数 A の値を得る。*/
```

```
s = "a";
int type = mcrcalc(s, &dpval, cstr, sizeof(cstr));
switch (type) {
  case 0: /* No output */
    break;
  case 3: /* Number */
    break;
  case 4: /* Text */
    break;
  case 30: /* Array */
    break;
  default:
    break;
}
```

18.7.3 ファイル名の一覧を表示

ファイル名を画面に表示し、該当するものを選ぶ。

【呼出し形式】

```
int FilSelect(int seltyp, const std::string& caption, const std::string& fctg,
              conststd::string& fmask, std::string* fname, size_t maxlen = 256L)
```

【入力引数】

seltyp　選択の種類（ダイアログを使用する場合にのみ有効)
- 0 = 呼び出し用　：　ダイアログの OK ボタンは「開く」と表示される。既存ファイルの選択および入力が可能。
- 1 = 書き込み用　：　ダイアログの OK ボタンは「保存」と表示される。既存ファイルの選択および新ファイル名の入力が可能。
- 2 = 書き込み用　：　ダイアログの OK ボタンは「ＯＫ」と表示される。既存ファイルの選択および新ファイル名の入力が可能。

caption　ダイアログのキャプション（ダイアログを使用する場合にのみ有効）
ダイアログの左上に表示される文字列を指定する。EUC-JP。
空文字列を与えるとデフォルトのキャプション " ファイルの選択 " を表示する。

fctg　ファイルの種類。たとえば "#MODEL#", "#SYMBOL#"。

fmask　ファイル名。
- ワイルドカード '*'　：　任意の文字列
- ワイルドカード '?'　：　任意の一文字

maxlen　ファイル名の最長バイト数。省略時は 256 バイト。この引数は将来除去する予定なので省略するのが良いでしょう。

【出力引数】

fname　選択されたファイル名を格納する std::string オブジェクトへのポインタ。
ディレクトリパスおよびファイル拡張子は ACAD.SET の fctg での定義と一致する部分は は含まない。たとえば ACAD.SET でのディレクトリが "D:/acad/files/"、拡張子が !.MDL! のとき以下のようになる。

D:/acad/files/ABC.MDL	->	ABC
D:/acad/files/subdir/ABC.MDL	->	subdir/ABC
D:/acad/files/ABC.PLT	->	ABC.PLT
C:/temp/ABC.MDL	->	C:/temp/ABC

フルパスを得るには AwFile::ToFullPath() メソッドを使います。

【戻り値】

0　：　正常に選択された。または 入力ファイル名にワイルドカードが含まれていない。

1　　：　ファイルが選択されなかった。
3　　：　ファイル名が maxlen を超える。

注意

(1)　戻り値が 1、3 のとき、メッセージ領域にエラーメッセージを表示している。
(2)　ファイルが選択されたとき、そのファイル名が入力されたものとして、セッションファイルに記録される。

例

```
/* デフォルトのキャプション、ファイル名が ABC で始まる既存のモデルファイルを選択する */
std::string fname;
if (FilSelect(0, "", "#MODEL#", "ABC*", &fname) != 0)
  return 1; /* ファイルが選択されなかった */
std::string path = AwFile::ToFullPath("#MODEL#", fname); /* フルパスにする */
if (path.empty()) {
  /* #MODEL# が ACAD.SET に定義されていない */
  /* 起動ディレクトリで拡張子なしのファイル名になっている */
  /* 正常としてもよい場合もある */
}
```

第 19 章　基本的な図形表示モジュール

この章の関数はデータベースアイテムおよびテンポラリアイテムを表示または消去するものではありません。座標値を指定して線分、円、円弧、自由曲線、文字列を表示または消去するものです。

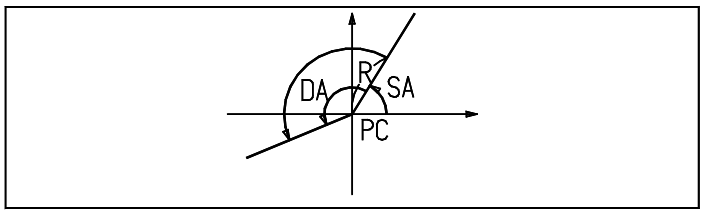
19.1 基本図形表示モジュール

● 関数一覧

関数名	機能
Dsparc	円弧を表示する。
Dsparc3	３点円弧を表示する。
Dspclr	カラーまたは消去モードを設定する。
Dspend	Dspinit で作成した、表示用のサブウインドウを削除する。
Dsperas	表示領域全体または指定領域を消去する。
Dspinit	図形表示の使用環境を設定する。
Dsplft	線種を設定する。
Dspline	線分を表示する。
Dsplwt	線幅を設定する。
Dspmark	マークを表示する。
Dspmsg	メッセージを表示する。
Dspmsgers	メッセージを消す。
Dspscrn	表示プレーンを選択する。
Dspspln	自由曲線を表示または消去する。
Dspspls	3 次 Bezier 曲線を表示または消去する。
Dsptext	文字列を表示する。
Dspwcs	補助座標系を設定する。
Dspzone	表示領域座標を得る。

19.1.1 円弧の表示・削除

円弧を表示または消去する。使用に先だって Dspinit 関数で環境設定しなければならない。



【呼出し形式】

```
void Dsparc(const FPOINT* pc, double r, double sa, double da)
```

【入力引数】

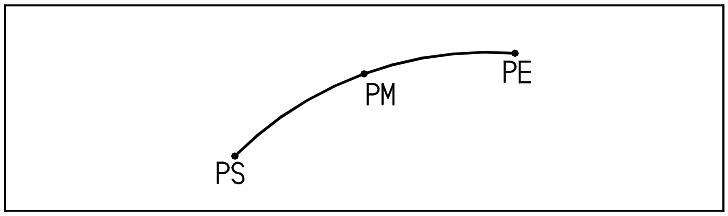
pc	円弧の中心点。以下座標は補助座標系で指定する。
r	円弧の半径
sa	円弧の始点角度（度）
da	円弧の内角（度）。 正 ： 反時計回り、負 ： 時計回り、360.0 ： 完全円。

例 . 中心点 (10,5)、半径 5 の円を表示する。

```
FPOINT pc = {10.0, 0.0};
Dsparc(&pc, 5.0, 0.0, 360.0);
```

19.1.2 円弧の表示・削除

３点円弧を表示または消去する。使用に先だって Dspinit 関数で環境設定しなければならない。



【呼出し形式】

```
void Dsparc3(const FPOINT* ps, const FPOINT* pm, const FPOINT* pe)
```

【入力引数】

ps	円弧の始点座標。以下座標は補助座標系で指定する。
pm	円弧の中間点座標
pe	円弧の終点座標

例

始点 (10,0)、通過点 (20,10)、終点 (30,0) の円弧を表示する。

```
FPOINT ps = {10.0, 0.0};
FPOINT pm = {20.0, 10.0};
```

```
FPOINT pe = {30.0, 0.0};
Dsparc3(&ps, &pm, &pe);
```

19.1.3 カラーまたは消去モードの設定

表示のカラーまたは消去モードを設定する。使用に先だって Dspinit 関数で環境設定しなければならない。

【呼出し形式】

```
void Dspclr(int iclr)
```

【入力引数】

iclr　　カラー番号
テンポラリ図形用スクリーン、グリッド用スクリーン、
ラバーバンド用スクリーンが選択されている時
　0　　：　以後の表示は消去となる
　1　　：　以後の表示は表示となる (default)
実図形用スクリーンが選択されている時
　0　　：　以後の表示は消去となる
　1-n　　：　以後の表示は指定されたカラーとなる (default==1)

19.1.4 表示用サブウィンドウの削除

Dspinit() で作成した表示用のサブウインドウを削除する。この関数はラスター座標系のときに有効。再度、図形表示関数を使用するには Dspinit 関数で環境設定しなければならない。

【呼出し形式】

```
void Dspend(void)
```

19.1.5 表示用サブウィンドウの消去

表示領域全体または指定領域を消去する。使用に先だって Dspinit 関数で環境設定しなければならない。この関数は補助座標系は考慮していない。絶対座標系で指定する。

【呼出し形式】

```
void Dsperas(int iflg, const FRECT* rect)
```

【入力引数】

iflg　　消去する部分の選択
　0　　：　表示領域全体を消去する。この場合、引数 rect は参照しない。
　　表示領域については Dspinit 関数を参照。
　1　　：　rect で指定された領域内を消去する
rect　　消去する領域の左下と右上座標

例

(1)　表示領域全体を消去する。

```
Dsperas(0, (FRECT *)NULL);
```

(2)　領域 (0,0)-(100,20) 内を消去する。

```
FRECT rect = {0.0, 0.0, 100.0, 20.0};
Dsperas(1, &rect);
```

19.1.6 図形表示関数の使用環境を設定

図形表示関数の使用環境を設定する。これは図形表示関数を使用するに先だって必ず一度は呼び出す必要がある。

【呼出し形式】

```
void Dspinit(int iscr, int iunit)
```

【入力引数】

iscr　プレーンの選択
- 0　：テンポラリ図形用プレーン
- 1　：実図形用プレーン
- 2　：グリッド用プレーン
- 3　：ラバーバンド用プレーン

iunit　座標系。表示領域はモジュール Dspzone で得ることができる。
- 0　：ラスター座標系
  メニューファイル ACADZON.MEN のキーワード Graphic zone で定義された座標系。
  これが選択されると ACADZON.MEN のウインドウ番号 #1 の大きさのサブウインドウが作成され、図形はサブウインドウの下になり見えなくなる。
  以下図形表示関数はこのサブウインドウに表示する。
  このサブウインドウの行列数は Menuzone 関数で得ることができる。
- 1　：データベース座標系
  アクティブビューポート内のピクチャの表示状態と同じ位置関係となる。

注意
ラスター座標系で使用した場合、最後にサブウインドウを削除するために Dspend() を呼び出す必要がある。Dspend () を呼出さない場合でもメインカテゴリのコマンドが要求されると、Dspinit() で作成したサブウインドは削除される。

例 . 実図形用スクリーンにデータベース座標系で表示する。

```
Dspinit(1, 1);
```

19.1.7 線種の設定

線種を設定する。使用に先だって Dspinit 関数で環境設定しなければならない。

【呼出し形式】

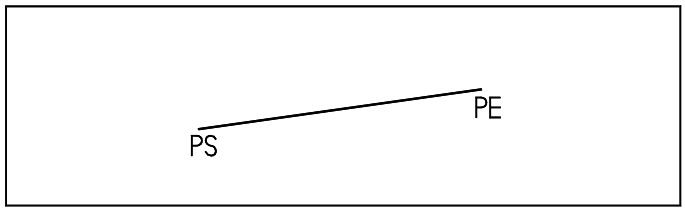
```
void Dsplft(int lft)
```

【入力引数】

lft　線種番号。1 - MAXITMLFT (default==1)

19.1.8 線分の表示・消去

線分を表示または消去する。使用に先だって Dspinit 関数で環境設定しなければならない。



【呼出し形式】

```
void Dspline(const FPOINT* ps, const FPOINT* pe)
```

【入力引数】

ps	線分の始点座標。以下座標は補助座標系で指定する。
pe	線分の終点座標

19.1.9 線幅の設定

線幅を設定する。使用に先だって Dspinit 関数で環境設定しなければならない。

【呼出し形式】

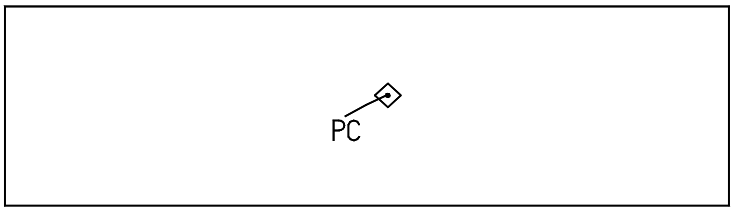
```
void Dsplwt(int lwt)
```

【入力引数】

lwt	線幅番号。1-MAXITMLWT (default==1)

19.1.10 マークの表示・消去

マークを表示または消去する。使用に先だって Dspinit 関数で環境設定しなければならない。



【呼出し形式】

```
void Dspmark(const FPOINT* pc, int mark)
```

【入力引数】

pc	マークの中心点。補助座標系で指定する。

mark　　マーク番号

番号	形	大きさ (画面上のミリ)
1	□	4
5	X	4
6	Y	4
14	・	0.5
15	＋	2
16	＊	2
17	○	2
18	X	2
101	◇	2

19.1.11 メッセージの表示

メッセージを表示する。この関数はラスター座標系のときに有効。使用に先だって Dspinit 関数で環境設定しなければならない。

【呼出し形式】

```
void Dspmsg(int row, int col, int msg, int type, const void* data)
```

【入力引数】

row　　行番号
col　　列番号
msg　　メッセージ番号。0 : メッセージなし
type　　データタイプ
　　メッセージの後に表示するデータのタイプ。
　　0　　: なし
　　1　　: short 型 (16 bits)
　　2　　: float 型
　　3　　: double 型
　　4　　: int 型
　　n*10　: char 型 (n は文字数)
data　　データ

19.1.12 メッセージの消去

メッセージを消す。この関数はラスター座標系のときに有効。使用に先だって Dspinit 関数で環境設定しなければならない。

【呼出し形式】

```
void Dspmsgers(int row, int col)
```

【入力引数】

row　　行番号。0 : 全ての行
col　　列番号。0 : 全ての列

19.1.13 プレーンの選択

プレーンを選択する。使用に先だって Dspinit 関数で環境設定しなければならない。

【呼出し形式】

```
void Dspscrn(int iscr)
```

【入力引数】

iscr　　プレーンの番号
　　0　：テンポラリ図形用プレーン
　　1　：実図形用プレーン
　　2　：グリッド用プレーン
　　3　：ラバーバンド用プレーン

19.1.14 自由曲線の表示・消去

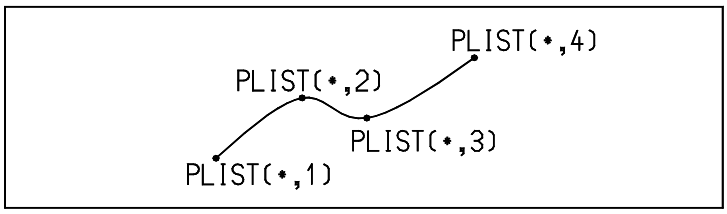
自由曲線を表示または消去する。使用に先だって Dspinit 関数で環境設定しなければならない。

【呼出し形式】

```
void Dspspln(const FPOINT plist[], int n, int cswt)
```

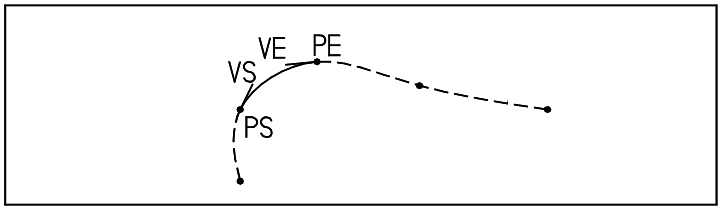
【入力引数】

plist　　**自由曲線の通過点列。**閉曲線の時、終点を含んでも (終点＝始点)、含まなくてもよい。
　　座標は補助座標系で指定する。
n　　**点数。最小は 3、最大はヘッダーファイル acadprm.h 内の MAXSPLPNT。**
cswt　　**開曲線か閉曲線かの指定。0 : 開曲線、1 : 閉曲線**



19.1.15 ３次 Bezier 曲線の表示・消去

3 次 Bezier 曲線を表示または消去する。使用に先だって Dspinit 関数で環境設定しなければならない。



【呼出し形式】

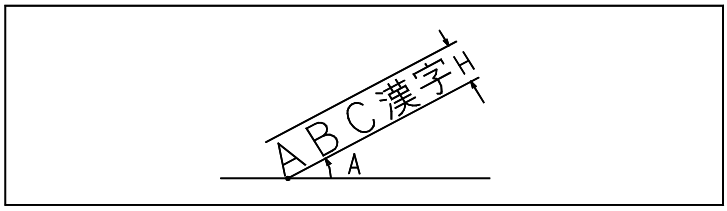
```
void Dspspls(const FPOINT* ps, const FPOINT* vs, const FPOINT* ve, const FPOINT* pe)
```

【入力引数】

ps	3 次 Bezier 曲線の始点。以下座標は補助座標系で指定する。
vs	3 次 Bezier 曲線の始点側のバーテックス点
ve	3 次 Bezier 曲線の終点側のバーテックス点
pe	3 次 Bezier 曲線の終点

19.1.16 文字列の表示・消去

文字列を表示または消去する。使用に先だって Dspinit 関数で環境設定しなければならない。



【呼出し形式】

```
void Dsptext(const FPOINT* ps, double h, double a, const char* text)
```

【入力引数】

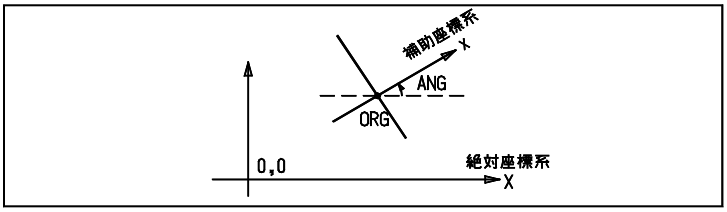
ps	文字列表示位置（左下）。補助座標系で指定する。
h	文字高さ
a	表示角度（度）
text	文字列 (Null-char terminated)。256 バイト以下

例 . 文字列 "ABC 漢字 " を左下座標 (10,5)、高さ 10、角度 30 度で表示する。

```
FPOINT ps = {10.0, 5.0};
dsptext(&ps, 10.0, 30.0, "ABC 漢字 ");
```

19.1.17 補助座標系を設定

補助座標系を設定する。使用に先だって Dspinit 関数で環境設定しなければならない。



【呼出し形式】

```
void Dspwcs(const FPOINT* org, double ang)
```

【入力引数】

org	補助座標原点。絶対座標系で指定する (default==0,0)
ang	補助座標 X 軸の角度（度）。絶対座標系で指定する (default==0)

注意
補助座標を設定後、次の関数は補助座標系で処理する。
Dsparc, Dsparc3, Dspline, Dspmark, Dspspln, Dspspls, Dsptext
次の関数は常に絶対座標系で処理する。
Dsperas, Dspzone

例
(1) 絶対座標 (10,20) を原点、X 軸の角を 30 度の補助座標を設定する。

```
FPOINT org = {10.0, 20.0};
Dspwcs(&org, 30.0);
```

(2) 絶対座標系に戻す。

```
FPOINT org = {0.0, 0.0};
Dspwcs(&org, 0.0);
```

19.1.18 表示領域座標を取得

表示領域座標を得る。Dspinit で指定した座標系に基づいた表示領域の座標を得る。使用に先だって Dspinit 関数で環境設定しなければならない。

【呼出し形式】

```
void Dspzone(FRECT *zone)
```

【出力引数】

zone　　表示領域の絶対座標
zone->pmin　　：　左下座標
zone->pmax　　：　右上座標

Appendix A　ユーザライブラリのビルド (Windows)

Advance CAD はユーザ用の共有ライブラリを動的リンクしています。ユーザは必要なユーザ関数を含む共有ライブラリをバインドしなおすだけです。つづいて Advance CAD を起動するだけで、ユーザ関数をテストできます。この方法は、Advance CAD 全体をリンクするのではなく、ユーザの共有ライブラリだけをリンクするのでリンク時間が短く、デバッグの効率が向上する利点があります。

- **プラットフォーム**
  Windows 7 Professional x64 edition (Service Pack 1 以上)

- **コンパイラ**
  Visual Studio 2010 C++ 日本語版

- **プログラム**
  Advance CAD は以下のファイルで構成されています。

acad.exe	(実行可能形式)
acadbc.dll	(基本・ダイナミックリンクライブラリ)
acadcore.dll	(システム・ダイナミックリンクライブラリ)
acadcontrol.dll	(コマンド制御・ダイナミックリンクライブラリ)
acadcui.dll	(コマンド・ダイナミックリンクライブラリ)
acadmdl.dll	(モデル・ダイナミックリンクライブラリ)
acadgr.dll	(グラフィック・ダイナミックリンクライブラリ)
acaddlg.dll	(ダイアログ・ダイナミックリンクライブラリ)
acaddlgcv.dll	(ダイアログ・ダイナミックリンクライブラリ)
acaddlgcv2.dll	(ダイアログ・ダイナミックリンクライブラリ)
acaddlglib.dll	(ダイアログ・ダイナミックリンクライブラリ)
acadstd.dll	(標準機能・ダイナミックリンクライブラリ)
acadsxf.dll	(SXF・ダイナミックリンクライブラリ)
acadxrf.dll	(モデル検索・ダイナミックリンクライブラリ)
acaddlgxrf.dll	(ダイアログ・ダイナミックリンクライブラリ)
acadcadam.dll	(CADAM・ダイナミックリンクライブラリ)
acadcatia.dll	(Catia・ダイナミックリンクライブラリ)
acaddxf.dll	(DXF&DWG・ダイナミックリンクライブラリ)
acadnc.dll	(NC・ダイナミックリンクライブラリ)
acadpid.dll	(P&ID シーケンス図・ダイナミックリンクライブラリ)
acadswi.dll	(SolidWorks I/F・ダイナミックリンクライブラリ)
sfc320.dll	(SXF 仕様 .sfc 用ライブラリ)
sxf320.dll	(SXF 仕様 .p21 用ライブラリ)
acadjpeg.dll	(JPEG・ダイナミックリンクライブラリ)
acadtiff.dll	(TIFF・ダイナミックリンクライブラリ)
acaduser.dll	(ユーザプログラミングインタフェース・ダイナミックリンクライブラリ)

- **ユーザ関数の組み込み**
  Advance CAD は acaduser.dll をダイナミックリンクしています。ユーザは必要なユーザ関数を含む acaduser.dll を作成し直すことによって関数を組み込むことができます。続いて Advance CAD を起動するだけでユーザ関数をテストすることができます。古いバージョンの dll は使用できませんので、必ず作成し直してください。

● **システムファイル**

以下のファイルは exe ディレクトリにあります。

変更してはならないファイル (EXE, DLL。acaduserdll を除く全て)

```
acad.exe
acadbc.dll      acadcore.dll   acadcontrol.dll acadcui.dll    acadmdl.dll
acadgr.dll      acaddlg.dll    acaddlglib.dll  acaddlgcv.dll  acaddlgcv2.dll
acaddlgxrf.dll  acadstd.dll    acadsxf.dll     acadxrf.dll    acadtiff.dll
acadcadam.dll   acadcatia.dll  acaddxf.dll     acadnc.dll     acadpid.dll
acadswi.dll     acadjpeg.dll   sfc320.dll      sxf320.dll
```

ユーザが再作成するファイル (DLL)

```
acaduser.dll
```

バージョン 12 から Advance CAD に必要なダイナミックリンクライブラリはすべて acad.exe と同じディレクトリに置きます。
これは Windows の標準的な方法で、以前のように dll ファイルのあるディレクトリを環境変数 PATH に追加する必要がありません。

以下のファイルは user ディレクトリにあります。

変更してはならないファイル (LIB)

```
acadbc.lib    acadcontrol.lib  acadcui.lib  acadmdl.lib  acadgr.lib  acadstd.lib
```

変更してはならないインクルードファイル

```
acaddef.h acadprm.h acadupi.h acadusr.h Aw*.h G2*.h G3*.h
```

非推奨関数のインクルードファイルとソースコード

```
acadlegacy.h acadlegacy1.cpp acadlegacy2.cpp
```

ユーザが再作成するファイル

```
acaduser.lib, acaduser.exp (nmake で作成される)
```

以下のフィルは sample/USER ディレクトリにあります。必要なファイルを user ディレクトリにコピーし、コピー後のファイルを修正します。

```
dspatch32.cpp dspatch64.cpp dspatch80.cpp dspatch88.cpp
usrcom.cpp usrmdm.cpp udbaccess.cpp
Makefile
```

● **コンパイル／リンク**

Advance CAD を停止してからコンパイル／リンクを実行してください。
Makefile を修正して、nmake とタイプすれば、コンパイルし、ダイナミックリンクライブラリ acaduser.dll をカレントディレクトリに作り直し、かつ dll を exe ディレクトリにコピーします。
Makefile の修正は、CSRCS にコンパイルするソースコードファイル名を追加するだけです。たとえば、udr01.cpp, udr02.cpp, udr03.cpp を追加する場合はつぎのように変更します。

```
CSRCS =¥
dspatch32.cpp dspatch64.cpp dspatch80.cpp dspatch88.cpp ¥
udbaccess.cpp usrcom.cpp usrmdm.cpp ¥
udr01.cpp udr02.cpp udr03.cpp
```

Advance CAD は 64bit アプリケーションです。ユーザ DLL は machine x64 でコンパイル、リンクします。以下に示すように dumpbin ツールを使って DLL のマシンタイプを調べることができます。

```
dumpbin /HEADERS acaduser.dll | findstr machine
8664 machine (x64)
```

● 実行

Advance CAD を実行し、ユーザ関数へ制御が渡ることを確認してください。
システムはダイナミックリンクライブラリを下記の順番で探しますので、ユーザ関数へ制御が渡らない場合は acaduser.dll が検索対象ディレクトリにあることを確認してください。

(1) 実行形式 (acad.exe) があるディレクトリ (デフォルトでは dll をここに置く)
(2) カレントディレクトリ (起動ディレクトリ)
(3) Windows システムディレクトリ (GetSystemDirectory 関数で取得できる)
(4) Windows ディレクトリ (GetWindowsDirectory 関数で取得できる)
(5) 環境変数 PATH に設定したディレクトリ

● 参考

Makefile

```
# AdvanceCAD Ver 20 (ITOCHU TECHNO-SOLUTIONS Corporation)
#
# Purpose : Make user DLL
#
!include <ntwin32.mak>

CFLAGS   = $(cflags) $(cdefs) $(cvarsdll) -DAcadUserIMPL -nologo /EHsc
CINCS    = -I.

EXEPATH  = ..¥exe
USERDLL  = acaduser.dll
EXPFILE  = acaduser.exp
IMPLIB   = acaduser.lib
DLLFLAGS = $(dlllflags)
DLLLIBS  = $(guilibsdll)
NETLIBS  =
ACADLIB  = acadbc.lib acadcontrol.lib acadcui.lib acadgr.lib ¥
           acadmdl.lib acadstd.lib
LIBS     = $(DLLLIBS) $(NETLIBS) $(ACADLIB)

CSRCS = ¥
dspatch32.cpp dspatch64.cpp dspatch80.cpp dspatch88.cpp ¥
udbaccess.cpp usrcom.cpp usrmdm.cpp ¥
acadlegacy1.cpp acadlegacy2.cpp

COBJS = $(CSRCS:.cpp=.obj)
all: dll copy
.cpp.obj:
        $(cc) $(CFLAGS) $(CINCS) $<

dll: $(USERDLL)

$(EXPFILE): $(COBJS)
        $(implib) -machine:$(CPU) -out:$(IMPLIB) -def: @<<
$(COBJS)
<<

$(USERDLL): $(COBJS) $(EXPFILE)
        $(link) $(DLLFLAGS) -out:$@ $(COBJS) $(LIBS)

copy:
        copy $(USERDLL) $(EXEPATH)

clean:
        del *.obj $(EXPFILE) $(IMPLIB) $(USERDLL)
# end of file
```

Appendix B　ヘッダーファイル

アプリケーションが公開しているヘッダーファイルは以下のものです。
以下の４つのヘッダーファイルは従来からのものです。

acaddef.h	**共通のデータ型宣言、マクロ定義**
acadprm.h	**共通定数宣言**
acadupi.h	**フック関数のプロトタイプ宣言**
acadusr.h	**公開関数のプロトタイプ宣言**

以下のヘッダーファイルとソースコードはバージョン 20 で追加したものです。

acadlegacy.h	**非推奨関数（古い公開関数）のプロトタイプ宣言**
acadlegacy*.cpp	**非推奨関数（古い公開関数）の実装**

バージョン 20 では主要な機能が C++ クラスでの実装になりました。これに伴い、クラスの機能と重複する関数の説明を削除しました。説明が削除された関数のプロトタイプ宣言は acadusr.h から acadlegacy.h へ移動しました。およそ 7 割の関数を移動しました。これらの古い関数を使用し続けるのではなく、新しいクラスの使用に置き換えることを勧めます。
これらの古い関数のうちで可能なものは実装をソースコード acadlegacy*.cpp に置きました。古い関数はクラスを使用して以前とほぼ同じ機能を実現するようになっています。関数からクラスへ移行する際の参考になるかもしれません。しかし今までのやり方が、オブジェクト指向プログラミングに変わった今でも最善とは限りません。あるデータを変更する今までのやり方は、データ取得関数でデータをローカルな配列にコピイし、その配列の内容を変更し、データ設定関数で変更後のデータを設定するといった手順をとりました。オブジェクト指向の実装でもそれと同等なことができますが、より効率的な方法があります。変更したいデータを持つオブジェクトを取得し、そのオブジェクトのメソッドを使ってデータを変更することです。この方法は簡単であるだけでなく、安全性が高い方法です。ローカルな配列は必要ないし、データの変更も適切なメソッドが行うからです。
acadlegacy*.cpp に実装がない古い関数もあります。それらは大きな変更がなされた部分で、古い関数を使うことが困難を伴います。パーマネントアイテムにアクセスする関数 DbUrd*、テンポラリアイテム操作関数 Tmpg* は古い ItmSR 構造体を使用しています。現在、アイテムを構成するサブレコードは AwSr クラス実装になりましたので、クラスを使用する方法が効率が良く安全です。ItmSR 構造体を使う古い関数を使うのは不適切です。幾何演算関数 gmu* や座標変換関数は DGEOM、DPOINT、DLINE、DARC、DBZC 構造体などを使用しています。それらは幾何クラス実装になりました。また、以前は曲線の種類ごとに異なっていた（線分は長さ、円弧は角度といったような）パラメータを曲線長に統一しました。幾何演算関数ではなく幾何クラスを使用する方法が効率が良く安全です。

ユーザのソースコードが新しいクラスを使用して書き換えられたなら acadlegacy.h に記述した関数は不要になります。そうなったら acadlegacy.h、acadlegacy*.cpp は削除してかまいません。

下記のクラスのヘッダーファイルはバージョン 20 で追加したものです。これらのヘッダーファイルの内容を変更してはいけません。また、これらのクラスを継承したサブクラスを定義しないでください。

Aw*.h	**本書で説明したクラスのヘッダーファイル (78)**
G2*.h	**本書で説明したクラスのヘッダーファイル (13)**
G3*.h	**本書で説明したクラスのヘッダーファイル (2)**

AwActiveList.h AwColorAssignment.h AwCompositeCreator.h AwCurveItemIterator.h
AwCurveOffsetter.h AwCurveOperator.h AwDimCreator.h AwDimText.h
AwDlo.h AwDloFrame.h AwDloList.h AwDloTitle.h
AwDloTitleDefinition.h AwDloTitleInstance.h AwDloTitleSet.h AwDloWindow.h
AwDrafParam.h AwDragLoader.h AwEucEncoder.h AwFile.h
AwFileIO.h AwGeomParam.h AwGridSet.h AwHighlightItems.h
AwHighlightPoints.h AwHighlightSet.h AwItem.h AwItemAttrTable.h
AwItemAttributes.h AwItemEditor.h AwItemExploder.h AwItemIdArray.h
AwItemIdsIterator.h AwItemListIterator.h AwMaskSet.h AwMassProperty.h
AwMiscParam.h AwModel.h AwModelTitle.h AwModelTitleSet.h
AwNoteCreator.h AwOffsetCalculator.h AwPenAssignment.h AwPermDbIterator.h
AwPermItemDb.h AwPickInfo.h AwPicture.h AwPictureList.h
AwSplineCreator.h AwSr.h AwSrAflParam.h AwSrAssociation.h
AwSrAttributes.h AwSrBezier.h AwSrCategory.h AwSrCircle.h
AwSrCurve.h AwSrDimension.h AwSrEoi.h AwSrGtFrame.h
AwSrLine.h AwSrMark.h AwSrNgText.h AwSrPlacement.h
AwSrPoint.h AwSrProperty.h AwSrScalar.h AwSrStart.h
AwSrText.h AwSrTimestamp.h AwSrXhtParam.h AwStringUtil.h
AwTempItem.h AwTempItemDb.h AwUserIO.h AwUtil.h
AwWcs.h AwWindow.h

G2Bezier.h G2Circle.h G2Conic.h G2Curve.h
G2Geom.h G2GeomArray.h G2Line.h G2Math.h
G2Matrix.h G2Point.h G2PointArray.h G2Rect.h
G2Vector.h

G3Point.h G3Rotation.h

Appendix C　ユーザコマンドの登録例

以下は、ユーザコマンドの登録に必要なファイルの例です。

コマンド定義ファイル	USERCMD.MEN
メニューファイル	USEROSM.MEN
メッセージファイル	MSG90.TXT
エラーメッセージファイル	ERR90.TXT
ユーザコマンドディスパッチャ	dspatch32.cpp
ユーザコマンドドライバー	udr01.cpp
コマンドハンドラ USER1	ucmd01.cpp
コマンドハンドラ USER2	ucmd02.cpp
コマンドハンドラ USER3	ucmd03.cpp
コマンドハンドラ USER4	ucmd04.cpp

filename : USERCMD.MEN

```
/
/ Advance CAD USER command list.
/
Command
/
/ + [dispatcher#, driver#, form#] !command_name!
/ command name :== [A-Z][A-Z0-9/_]*
/
/ [ 32, n, n] User defined command
/ [ 48, n, n] User defined command modifier
/
/ User commands
+ [32, 1, 0] !USER!
+ [32, 1, 1] !USER1!
+ [32, 1, 2] !USER2!
+ [32, 1, 3] !USER3!
+ [32, 1, 4] !USER4!
/
/ User command modifiers
+ [48, 1, 1] !OPTDUP!
+ [48, 1, 2] !OPTOFF!
/
/ End of file
```

filename : USEROSM.MEN

```
/
/ Advance CAD USER On screen menu definition
/
/ Menu [menu#,category#,screen#,colour#]
/
/ + <row#,col#> "text" !command! [menu1#,menu2#,colour#]
/ L <row#,col#> "text" !letter!  [menu1#,menu2#,colour#]
/ T <row#,col#> "text" !text!    [menu1#,menu2#,colour#]
/ N <row#,col#> "text" [menu1#,menu2#,colour#,type#,value]
/        type 0=scalar, 1=Mark#, 2=colour#, 3=key#
/
Menu [user_cmd,  1,  4,  1]
+ < 1, 1> "ユーザコマンド" !USER!   [user_cmd, user_mdf, 3]
+ < 3, 1> "ユーザ１"       !USER1!  [user_cmd, user_mdf, 4]
+ < 5, 1> "ユーザ２"       !USER2!  [user_cmd, user_mdf, 4]
+ < 7, 1> "ユーザ３"       !USER3!  [user_cmd, user_mdf, 4]
+ < 9, 1> "ユーザ４"       !USER4!  [user_cmd, user_mdf, 4]
/
Menu        [user_mdf,  1, 17,  1]
+ <10, 1> "  複製－ＯＮＮ" !OPTDUP!
+ <11, 1> "  複製－ＯＦＦ" !OPTOFF!
/
/ End of file
```

filename : MSG90.TXT

```
/ Filename : MSG90.TXT
/
/ ACAD message file for user command.
/
/        Message number        9000000 - 9999999
/
+ (9000000) "〈ＣＥ〉または〈ＢＳ〉を入力"
+ (9000001) "ボックスサイズの変更／ボックスの配置位置を指示"
+ (9000002) "移動するラインアイテムの選択／移動ベクトルの変更"
+ (9000003) "移動するストリングアイテムの選択／移動ベクトルの変更"
/
+ (9000010) "ボックスサイズ"
+ (9000011) "移動量　Ｘ"
+ (9000012) "移動量　Ｙ"
+ (9000013) "複製モードー ＯＦＦ"
+ (9000014) "複製モードー ＯＮ"
/
/ End of file
```

filename : ERR90.TXT

```
/ Filename : ERR90.TXT
/
/ ACAD error message file for user command.
/
/        Error number        9000000 - 9999999
/
+ (9000000) "ボックスサイズが有効でない。"
+ (9000001) "テンポラリポイントがえられない。"
```

```
+ (9000002) "アイテムが選択できない。"
+ (9000003) "アイテムがラインアイテムでない。"
+ (9000004) "ベクトルの長さが０である。"
+ (9000005) "選択されたアイテムがストリングアイテムでない。"
/
/ End of file
```

filename : dspatch32.cpp

```
/*
  Filename : dspatch32.cpp
  Category : User dispatcher
*/

#include "acaddef.h"
#include "acadprm.h"
#include "acadusr.h"
#include "acadupi.h"

/*************************************************************************
Purpose
     Example of user dispatcher
Inputs
     token  Token
Outputs
     None
*/
void dspatch32(TOKEN* token) {
  switch (Idriver(1)) {
  case 1:
    udr01(Iformat(1), token);
    break;
#if 0
  case 2:
    udr02(Iformat(1), token);
    break;
  case 3:
    udr03(Iformat(1), token);
    break;
#endif
  default:
    break;
  }
} /* dspatch32 */
```

filename : udr01.cpp

```
#include "acaddef.h"
#include "acadprm.h"
#include "acadusr.h"
#include "acadupi.h"

/*************************************************************************
  Purpose
      Example of user driver #1
  Inputs
      token   Token
  Outputs
      None
```

```
*/
void udr01(int form, TOKEN* token) {
  switch (form) {
  case 1:
    ucmd01(token);
    break;
  case 2:
    ucmd02(token);
    break;
  case 3:
    ucmd03(token);
    break;
  case 4:
    ucmd04(token);
    break;
  default:
    break;
  }
} /* udr01 */
```

filename : ucmd01.cpp

```
#include "acaddef.h"
#include "acadprm.h"
#include "acadusr.h"
#include "acadupi.h"

/**
 * USER1 command handler
 *   Create two line items and one string item.
 * Command Syntax
 *   USER1 <CE>
 */
void ucmd01(TOKEN* token) {
  int ier = 0;

  switch (token->typ) {
  case TknCMD:
  {
    G2Point dpnts[] = { G2Point( 100.0, 0.0), G2Point(0.0,  100.0),
                        G2Point(-100.0, 0.0), G2Point(0.0, -100.0),
                        G2Point( 100.0, 0.0) };
    AwTempItemDb* pTempItemDb = GetActiveModel()->GetTempItemDb();
    pTempItemDb->StoreItems();
    pTempItemDb->Init();
    // Create Two line items.
    for (int j = 0; j < 2; ++j) {
      AwTempItem* pTempItem = pTempItemDb->OpenItem();
      if (pTempItem == NULL)
        continue;
      pTempItem->SetType(AwItem::LINE);
      pTempItem->AddSr(AwSrStart(dpnts[j]));
      pTempItem->AddSr(AwSrLine(dpnts[j+1]));
      pTempItem->Close();
    }
    // Create the string item.
    AwTempItem* pTempItem = pTempItemDb->OpenItem();
    if (pTempItem) {
     pTempItem->SetType(AwItem::STRING)
```

```
      pTempItem->AddSr(AwSrStart(dpnts[2]));
      for (int j = 3; j < 5; ++j)
        pTempItem->AddSr(AwSrLine(dpnts[j]));
      pTempItem->Close();
    }
  }
    break;
  case TknBSP:
    if (GetActiveModel()->GetTempItemDb()->RemoveLastItem()) {
       Vieerase(0, TEMPSCREEN);
       rpttmppln();
    } else {
      ier = 100;
    }
    break;
  case TknEOC:
    GetActiveModel()->GetTempItemDb()->StoreItems();
    GetActiveModel()->GetTempItemDb()->Init();
    break;
 case TknEXIT:
    GetActiveModel()->GetTempItemDb()->Init();
    return;
 default:
    ier = 100;
    break;
  }
  if (ier)
    Errorb(); /* Ring the error bell. */
  Opmsgcode(1, 9000000); /* Display the operation message. */
}
```

filename : ucmd02.cpp

```
#include "acaddef.h"
#include "acadprm.h"
#include "acadusr.h"
#include "acadupi.h"

/**
 * USER2 command handler
 *   Create the box string item at the specified position.
 * Command Syntax
 *   USER2 [[s] P]+ <CE>
 *           s        Box size.
 *           P        Lower-left corner of the box.
 */
void ucmd02(TOKEN* token) {
  static double boxsize;
  DPOINT p;
  int ier = 0;

  switch (token->typ) {
  case TknCMD:
    GetActiveModel()->GetTempItemDb()->StoreItems();
    GetActiveModel()->GetTempItemDb()->Init();
    // Initialize the default box size./
    boxsize = 10.0;
    // Display the current box size on the command status area.
    Mesageras(MSGZONE, 0, 0);
```

```
  Mesagdisp(MZONECOLOR1, 1, 1, 9000010, 3, &boxsize);
  break;
case TknCOD: /* Coordinate */
case TknDIG: /* Digitize */
case TknPNT: /* Temporary Point */
  if (IdentPoint(token, &p) == IDENT_SUCCESS) {
    // Set the corner points of the box.
    G2Point boxcnr[4];
    boxcnr[0].x = p.x + boxsize;
    boxcnr[0].y = p.y;
    boxcnr[1].x = p.x + boxsize;
    boxcnr[1].y = p.y + boxsize;
    boxcnr[2].x = p.x;
    boxcnr[2].y = p.y + boxsize;
    boxcnr[3].x = p.x;
    boxcnr[3].y = p.y;
    // Create the temporary string item.
   AwTempItem* pTempItem = GetActiveModel()->GetTempItemDb()->OpenItem();
    if (pTempItem == NULL) {
      ier = 1;
    } else {
      pTempItem->SetType(AwItem::STRING);
      pTempItem->SetClass(4); // Change the class number.
      pTempItem->AddSr(AwSrStart(boxcnr[3]));
      for (int j = 0; j < 4; ++j)
        pTempItem->AddSr(AwSrLine(boxcnr[j]));
      pTempItem->Close();
    }
  }
  break;
case TknSCL:
  if (sc < 1.0) {
    ier = 1;
  } else {
    /* Display the current box size on the command status area. */
    boxsize = sc;
    Mesageras(MSGZONE, 0, 0);
    Mesagdisp(MZONECOLOR1, 1, 1, 9000010, 3, &boxsize);
  }
  break;
case TknBSP: /* Delete the last temporary item. */
  if (GetActiveModel()->GetTempItemDb()->RemoveLastItem()) {
     Vieerase(0, TEMPSCREEN);
     rpttmppln();
  } else {
    ier = 100;
  }
  break;
case TknEOC:
  GetActiveModel()->GetTempItemDb()->StoreItems();
  GetActiveModel()->GetTempItemDb()->Init();
  break;
case TknEXIT:
  GetActiveModel()->GetTempItemDb()->Init();
  return;
default:
  ier = 100;
  break;
}
/* Handle the error. */
```

```
  if (ier) {
    if ( 0 < ier && ier < 100) {
      Errorcode(ier + 9000000 - 1);
    } else {
      Errorb(); /* Ring the error bell */
    }
  }
  /* Display the operation message. */
  Opmsgcode(1, 9000001);
}
```

filename : ucmd03.cpp

```
#include "acaddef.h"
#include "acadprm.h"
#include "acadusr.h"
#include "acadupi.h"

/**
 * USER3 command handler
 *   Move the selected line item by the specified vector.
 * Command Syntax
 *    USER3 [[vec] IS]+ <CE>
 *           vec       Vector to move line items.
 *           IS        Ident the line item to be moved.
 */
void ucmd03(TOKEN* token) {
  static G2Point vec;

  int ier = 0;

  switch (token->typ) {
  case TknCMD:
    GetActiveModel()->GetTempItemDb()->StoreItems();
    GetActiveModel()->GetTempItemDb()->Init();
    vec.Set(10.0, 10.0);
    /* Display the current vector on the command status area. */
    Mesageras(MSGZONE, 0, 0);
    Mesagdisp(MZONECOLOR1, 1, 1, 9000011, 3, &vec.x);
    Mesagdisp(MZONECOLOR1, 2, 1, 9000012, 3, &vec.y);
    break;
  case TknDIG:
  {
    AwTempItemDb* pTempItemDb = GetActiveModel()->GetTempItemDb();
    pTempItemDb->StoreItems();
    pTempItemDb->Init();
    // Ident the item.
    const AwItem* pItem = PickItem(CMDCTG, *token, 1);
    if (pItem == 0) {
      ier = 3;
      break;
    } else if (pItem->GetType() != AwItem::LINE) {
      ier = 4;
      break;
    }
    // Move the item by the current vector.
    AwTempItem* pTempItem = pTempItemDb->OpenItem(pItem->GetItemId(), 0, 0);
    for (int i = 0; i < pTempItem->GetSrCount(); ++i) {
      AwSr* pSr = pTempItem->GetSrAt(i);
```

```
        if (pSr->GetDataType() == AwSr::EOI)
          break;
        pSr->Translate(vec);
      }
    }
      break;
    case TknVEC: /* Vector */
      if (token->pnt.x == 0.0 && token->pnt.y == 0.0) {
        ier = 5;
      } else {
        vec.Set(token->pnt.x, token->pnt.y);
        /* Display the current vector on the command status area. */
        Mesageras(MSGZONE, 0, 0);
        Mesagdisp(MZONECOLOR1, 1, 1, 9000011, 3, &vec.x);
        Mesagdisp(MZONECOLOR1, 2, 1, 9000012, 3, &vec.y);
      }
      break;
    case TknBSP: /* Delete the last temporary item. */
      if (GetActiveModel()->GetTempItemDb()->RemoveLastItem()) {
         Vieerase(0, TEMPSCREEN);
         rpttmppln();
      } else {
        ier = 100;
      }
      break;
    case TknEOC:
      GetActiveModel()->GetTempItemDb()->StoreItems();
      GetActiveModel()->GetTempItemDb()->Init();
      break;
    case TknEXIT:
      GetActiveModel()->GetTempItemDb()->Init();
      return;
    default:
      ier = 100;
      break;
    }

    /* Handle the error. */
    if (ier) {
      if (0 < ier && ier < 100) {
        Errorcode(ier + 9000000 - 1);
      } else {
        Errorb(); /* Ring the error bell */
      }
    }

    /* Display the operation message. */
    Opmsgcode(1, 9000002);
}
```

filename : ucmd04.cpp

```
#include "acaddef.h"
#include "acadprm.h"
#include "acadusr.h"
#include "acadupi.h"

/**
 * USER4 command handler
 *        Move the selected string items by the specified vector.
```

```
 * Command Syntax
 *        USER4 [[{OPTDUP, OPTOFF}] [vec] lSauto]+
 *              OPTDUP        Duplication mode on.
 *              OPTOFF        Duplication mode off.
 *              vec           Vector to move string items.
 *              lSauto        Ident the string items to be moved.
 */
void ucmd04(TOKEN* token) {
  static int keydup;
  static G2Point vec;

  int ier = 0;

  switch (token->typ) {
  case TknCMD: /* Command */
    GetActiveModel()->GetTempItemDb()->StoreItems();
    GetActiveModel()->GetTempItemDb()->Init();
    vec.Set(10.0, 10.0);
    keydup = 0;
    /* Display the current parameters on the command status area. */
    Mesageras(MSGZONE, 0, 0);
    Mesagdisp(MZONECOLOR1, 1, 1, 9000011, 3, &vec.x);
    Mesagdisp(MZONECOLOR1, 2, 1, 9000012, 3, &vec.y);
    Mesagdisp(MZONECOLOR2, 3, 1, (keydup == 0) ? 9000013 : 9000014, 0, (void *)NULL);
    break;
  case TknMDF: /* Modifier */
    if (token->cid[0] == 48 && token->cid[1] == 1 && (token->cid[2] == 1 || token->cid[2] == 2)) {
      /* OPTDUP [48, 1, 1] Duplication mode. */
      /* OPTOFF [48, 1, 2] No duplication mode. */
      keydup = (token->cid[2] == 1) ? 1 : 0;
      Mesageras(MSGZONE, 3, 0);
      Mesagdisp(MZONECOLOR2, 3, 1, (keydup == 0) ? 9000013 : 9000014, 0, (const void *)NULL);
    } else {
      ier = 100;
    }
    cmdmdfcla(1);
    break;
  case TknMZN: /* Message Zone */
    if (pnt->x == 1 && pnt->y == 3) { /* Change the duplication mode */
      keydup = (keydup == 0) ? 1 : 0;
      Mesageras(MSGZONE, 3, 0);
      Mesagdisp(MZONECOLOR2, 3, 1, (keydup == 0) ? 9000013 : 9000014, 0, (void *)NULL);
    } else {
      ier = 100;
    }
    break;
  case TknDIG: /* Digitize */
  case TknITM: /* Item Selection */
  {
    /* Ident the items. */
    if (IdentItems(1, token, 0) != IDENT_SUCCESS)
      break;
    /* Move the items. */
    AwTempItemDb* pTempItemDb = GetActiveModel()->GetTempItemDb();
    pTempItemDb->StoreItems();
    pTempItemDb->Init();
    int nstritm = 0; /* The number of the string items. */
    AwItemIdsIterator iterator(GetActiveModel()->GetPermItemDb(),
                               GetHighlightSet(1)->GetHlItems()->GetItemIds());
    while (iterator.HasNext()) {
      const AwItem* pItem = iterator.NextItem();
```

```
        if (pItem->GetType() != AwItem::STRING)
          continue;
        /* Move the item by the current vector. */
        AwTempItem* pTempItem = pTempItemDb->OpenItem(pItem->GetItemId(), keydup, 0);
        for (int i = 0; i < pTempItem->GetSrCount(); ++i) {
          AwSr* pSr = pTempItem->GetSrAt(i);
          if (pSr->GetDataType() == AwSr::EOI)
            break;
          pSr->Translate(vec);
        }
        pTempItem->Close();
        ++nstritm;
      }
      if (nstritm <= 0)
        ier = 6;
      pTempItemDb->StoreItems();
      pTempItemDb->Init();
      token->typ = TknCMD;
      (void)IdentItems(1, token, 0);
    }
      break;
    case TknVEC: /* Vector */
      if (token->pnt.x == 0.0 && token->pnt.y == 0.0) {
        ier = 5;
      } else {
        vec.Set(token->pnt.x, token->pnt.y);
        /* Display the current vector on the command status area. */
        Mesageras(MSGZONE, 1, 0);
        Mesagdisp(MZONECOLOR1, 1, 1, 9000011, 3, &vec.x);
        Mesageras(MSGZONE, 2, 0);
        Mesagdisp(MZONECOLOR1, 2, 1, 9000012, 3, &vec.y);
      }
      break;
    case TknEOC:
      break;
    case TknEXIT:
      return;
    default:
      ier = 100;
      break;
    } /* End of switch (token->typ) */

    /* Handle the error. */
    if (ier) {
      if (0 < ier && ier < 100) {
        Errorcode(ier + 9000000 - 1);
      } else {
        Errorb(); /* Ring the error bell */
      }
    }
    /* Display the operation message. */
    Opmsgcode(1, 9000003)
  }
```

Appendix D　プロセス間通信 (Windows)

D.1　概要

Advance CAD とユーザアプリケーションとの通信手段として、Windows の共有メモリを利用することができます。この機能を使用してユーザアプリケーションから Advance CAD へコマンドやパラメータを送信すると、ユーザアプリケーションから Advance CAD を制御することができます。また逆に、Advance CAD からユーザアプリケーションへデータを送信することもできます。この機能を使用するには、Windows の共有メモリの知識が必要です。

共有メモリを使用する通信は、次の手順で行います。

● 送信側

(1) ファイルマッピングオブジェクトを作成する（またはオープンする）。
(2) ファイルマッピングオブジェクトをメモリにマップする。
(3) マップされたメモリにデータを書き込む。
(4) 書き込んだことを受信側へ通知する（つまり、メッセージを送信する）。
(5) 受信側からの応答を待つ（メッセージが通知される）。
(6) 通信したいデータがなくなるまで、(3)、(4)、(5) を繰り返す。
(7) メモリにマップされたファイルマッピングオブジェクトをアンマップする。
(8) ファイルマッピングオブジェクトをクローズする。

● 受信側

(1) ファイルマッピングオブジェクトを作成する（またはオープンする）。
(2) ファイルマッピングオブジェクトをメモリにマップする。
(3) 送信側からの通知を待つ（メッセージが通知される）。
(4) マップされたメモリからデータを読み込む。
(5) 読み込んだことを送信側へ通知する（つまり、メッセージを送信する）。
(6) 受信したい間、(3)、(4)、(5) を繰り返す。
(7) メモリにマップされたファイルマッピングオブジェクトをアンマップする。
(8) ファイルマッピングオブジェクトをクローズする。

共有メモリはファイルマッピングオブジェクト名によって識別されます。通信しようとするアプリケーション同士はお互いに使用するファイルマッピングオブジェクトの名前を取り決めておかなければなりません。

また Advance CAD は他のユーザアプリケーションと通信するために 0xD000 からのメッセージ番号を使用します。プログラミングインターフェイスを使用する場合、メッセージ番号が同じにならないように注意してください。0xE000 からのメッセージ番号を使用することを推奨します。

D.2 ユーザアプリケーションから Advance CAD への送信

- **ファイルマッピングオブジェクト名**
  Advance CAD 起動時にコマンド引数として与えます。

  ```
  % acad -nFileMappingObjectName

          FileMappingObjectName : ファイルマッピングオブジェクト名
  ```

  送信側のアプリケーションは、このファイルマッピングオブジェクト名を使用して通信を行います。

- **Advance CAD の処理**
  Advance CAD が受信できるデータタイプは文字列で、一度に受信できる文字数は 80 バイト以下です。
  80 バイトを越えるとすべて無視されます。文字列の内容は、Advance CAD のコマンドストリームでなければなりません。Advance CAD は、指定された名前を持つ共有メモリにデータが書き込まれたことを通知するメッセージを受け取ると、共有メモリからデータを読み取ります。そして、それがキーボードから入力されたコマンドストリームとみなして解釈、処理します。したがって、この場合、Advance CAD 側はなんら変更する必要はありません。

- **通信側アプリケーションの処理**
  通信側アプリケーションは作成しなければなりません。作成する通信アプリケーションが行わなければならないことを、プログラム例で示します。プログラムの例は全体ではなく一部分です。また、エラーチェックなどを省略しています。プログラムの流れだけを確認してください。

```
#include <windows.h>
              :
              :
#define WM_ACADMSG 0xD000               /* Advnace CAD へ通知するメッセージの番号 */
              :
              :
static char lpszMapName[256];           /* ファイルマッピングオブジェクト名 */
              :
              :
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  static HANDLE hMapFile = NULL;
  static LPVOID lpMapped = NULL;
  static BOOL   bSendOk  = TRUE;
              :
              :
 switch (message) {
 case WM_CREATE:
   /* ファイルマッピングオブジェクトを作成する。既に存在する場合はオープンする。*/
   hMapFile = CreateFileMapping(INVALID_HANDLE_VALUE, NULL, PAGE_READWRITE,
                                0, 81, lpszMapName);
   if(hMapFile == NULL) {
     /* ここでエラー処理を行う */
   }

   /* ファイルマッピングオブジェクトをメモリにマップする。*/
   lpMapped = MapViewOfFile(hMapFile, FILE_MAP_WRITE, 0, 0, 0);
```

```
    if(lpMapped == NULL){
      /* ここでエラー処理を行う */
    }
    break;
  case WM_LBUTTONDOWN:
  case WM_MBUTTONDOWN:
  case WM_RBUTTONDOWN:
    /* データを共有メモリに書き込み、Advance CAD へメッセージを送信する。*/
    if(hMapFile == NULL || lpMapped == NULL) break;
    if(bSendOk) {
      strcpy((char *)lpMapped, "LBP <0,0> <100,100> <CE>");
      bSendOk = FALSE;
      PostMessage(HWND_BROADCAST, WM_ACADMSG, (WPARAM)hWnd, 0L);
    }
    break;
              :
              :
  case WM_ACADMSG:
    if((HWND)wParam == hWnd || (int)lParam == 0L) {
      /* 自分が発行したメッセージを受信したので、ここでは何もしない。*/
    } else {
      /* Advance CAD が発行したメッセージを受信したので、次のメッセージを送信できる。*/
      bSendOk = TRUE;
    }
    break;
              :
              :
  case WM_DESTROY:
    /* ファイルマッピングオブジェクトをアンマップし、クローズする。*/
    if(lpMapped != NULL)
      UnmapViewOfFile(lpMapped);
    if(hMapFile != NULL)
      CloseHandle(hMapFile);
      PostQuitMessage(0);
    break;
  default:
    return DefWindowProc(hWnd, message, wParam, lParam);
  }
  return 0;
}
```

D.3 Advance CAD からユーザアプリケーションへの送信

Advance CAD からユーザアプリケーションへの送信には、一部制限がありますが、上記で述べた共有メモリや Windows がサポートするプロセス間通信の機能を使用することができます。

Advance CAD 側では、プログラミングインタフェースを使って、送信する機能を追加します。これに対応した受信側アプリケーションを作成します。例として共有メモリを使用したプログラムを添付しますので、参考にしてください。

- Advance CAD 側

  ソースコード　:dspatch32.cpp userprop.cpp
  処理内容　: レジスタの内容、マクロ変数の内容を文字列に変更して、共有メモリに書き込む。

● 受信側アプリケーション

ソースコード　　:rec.c

処理内容　　：共有メモリの内容を読み出し、表示する。

filename : dspatch32.cpp

```
/*
  Filename : dspatch32.cpp
  Category : User dispatcher
*/

#include <stdio.h>
#include "acaddef.h"
#include "acadprm.h"
#include "acadupi.h"
#include "acadusr.h"

/* External Functions */
int xpropopen(void);
void xpropsend(char * data);

/* Static Functions */
static void udr02(int form, TOKEN *token);
static void ucmd21(TOKEN *token);

/*****************************************************************************
   Purpose
        Example of user dispatcher
   Inputs
        token   Token
   Outputs
        None
*/
void dspatch32(TOKEN *token)
{
  switch (Idriver(1)) {
  case 2:
    udr02(Iformat(1), token);
    break;
  default:
    break;
  }
} /* dspatch32 */

/*****************************************************************************
   Purpose
        Example of user driver #2
   Inputs
        form    Form number
        token   Token
   Outputs
        None
*/
static void udr02(int form, TOKEN *token)
{
  switch (form) {
  case 1:
    ucmd21(token);
    break;
```

```
  default:
    break;
  }
} /* udr02 */

/*************************************************************************
  Purpose
        Process UREG Command
*/
static void ucmd21(TOKEN *token)
{
  static int ipropini = 0;
  int ier = 0;

  switch (token->typ) {
  case TknCMD: /* Command */
   if (ipropini == 0) {
     ier = xpropopen();
     if (ier == 0)
       ipropini = 1;
   }
   cmdmdfcla(1);
   break;

  case TknTXT: /* Text */
    {
      int type;
      double dpval;
      char cstr[81];
      type = mcrcalc(token->txt, &dpval, cstr, sizeof(cstr));
      if (type == 3 || type == 4) {
        if (type == 3)
           sprintf(cstr, "%g", dpval);
        Mesageras(MSGZONE, 2, 1);
        Mesagdisp(MZONECOLOR1, 2, 1, 0, 10 * strlen(cstr), cstr);
        if (ipropini)
          xpropsend(cstr);
      } else if (type < 0) {
        ier = 2;
      }
    }
    break;

  case TknEOC: /* CE */
    break;

  case TknEIXT: /* Exit */
    return;

  default:
    ier = 100;
    break;
  } /* End of switch */

  if (ier)
    Errorb();
} /* ucmd21 */
```

filename : userprop.cpp

```
/*
  Filename : userprop.cpp
  File-mapping object Sample Code
  Use with Advance CAD Programming Interface
*/

#include <stdio.h>
#include <windows.h>
#include "acadusr.h"

#define WM_USERMSG 0xE000       /* Message # to communicate Advance CAD
                                   Programming Interface with User
                                   Application */

static HANDLE hMapFile = NULL;  /* Handle of the File-Mapping Object */
static LPVOID lpMapped = NULL;  /* Starting Address of the Mapped View */
static char   *MyMmap  = "__ACADMMAP_USERSEND";
                                /* Name of the File-Mapping Object */

/* Create and Map File-Mapping Object for User Application */
int xpropopen(void)
{
  hMapFile = CreateFileMapping(INVALID_HANDLE_VALUE, NULL, PAGE_READWRITE,
                               0, 1024, MyMmap);
  if(hMapFile == NULL)
    return 1;
  lpMapped = MapViewOfFile(hMapFile, FILE_MAP_WRITE, 0, 0, 0);
  if(lpMapped == NULL) {
    CloseHandle(hMapFile);
    return 1;
  }
  return 0;
}

/* Send Data from Advance CAD to User Application */
void xpropsend(char *data)
{
  if(lpMapped == NULL) return;
  euc2sjis(data, data, strlen(data));
  strcpy(lpMapped, data);
  PostMessage(HWND_BROADCAST, WM_USERMSG, 0L, 0L);
}
```

filename : rec.c

```
/*
  Filename: rec.c
  Advance CAD File-Mapping Object Sample Program
*/
#include <windows.h>
#include <string.h>
#include <stdio.h>

#define WM_USERMSG 0xE000       /* Message # to communicate Advance CAD
                                   Programming Interface with User
                                   Application */

LRESULT CALLBACK WindowFunc(HWND, UINT, WPARAM, LPARAM);

static char szWinName[] = "ACADTEST";            /* Name of Window Class */
```

```
static char MyMmap[]    = "__ACADMMAP_USERSEND"; /* Name of the File-Mapping Object */

int WINAPI WinMain(HINSTANCE hInst, HINSTANCE hPrevInst,
                   LPSTR lpszArgs, int nWinMode)
{
    HWND hWnd;
    MSG msg;
    WNDCLASS wcl;

    /* define Window Class */
    wcl.hInstance     = hInst;          /* Handle of current instance */
    wcl.lpszClassName = szWinName;      /* Name of window class */
    wcl.lpfnWndProc   = WindowFunc;     /* Window procedure */
    wcl.style         = 0;              /* Default style */

    wcl.hIcon         = LoadIcon(NULL, IDI_APPLICATION); /* Icon style */
    wcl.hCursor       = LoadCursor(NULL, IDC_ARROW);     /* Cursor style */
    wcl.lpszMenuName  = 0;              /* No menu */

    wcl.cbClsExtra    = 0;              /* Additional Information */
    wcl.cbWndExtra    = 0;              /* None */

    /* set window to light-gray */
    wcl.hbrBackground = GetStockObject(LTGRAY_BRUSH);

    /* register window class */
    if (!RegisterClass(&wcl)) return 0;

    /* create window */
    hWnd = CreateWindow(
        szWinName,              /* Window Class Name */
        "WindowsNT Skeleton",   /* Title */
        WS_OVERLAPPEDWINDOW,    /* Window Style - Normal */
        100,                    /* x */
        100,                    /* y */
        100,                    /* width */
        100,                    /* height */
        NULL,                   /* Handle of Parent - None */
        NULL,                   /* No Menu */
        hInst,                  /* Handle of program's this instance */
        NULL                    /* End of parameter */
        );

    /* visualize window */
    ShowWindow(hWnd, nWinMode);
    UpdateWindow(hWnd);

    /* message loop */
    while (GetMessage(&msg, NULL, 0, 0)) {
        TranslateMessage(&msg); /* enable keyboard */
        DispatchMessage(&msg);  /* return to WindowsNT */
    }
    return msg.wParam;
}

/*
    This procedure is called by WindowsNT, passes a message
    from the message queue.
*/
LRESULT CALLBACK WindowFunc(HWND hWnd, UINT message, WPARAM wParam,
                            LPARAM lParam)
{
```

```
    static HANDLE hMapFile = NULL; /* Handle of the File-Mapping Object */
    static LPVOID lpMapped = NULL; /* Starting Address of the Mapped View */
    HDC hdc;

    switch (message) {
    case WM_CREATE:
        hMapFile = CreateFileMapping(INVALID_HANDLE_VALUE, NULL, PAGE_READWRITE,
                                     0, 1024, MyMmap);
        if(hMapFile == NULL) break;
        lpMapped = MapViewOfFile(hMapFile, FILE_MAP_WRITE, 0, 0, 0);
        if(lpMapped == NULL) CloseHandle(hMapFile);
        SetWindowPos(hWnd,HWND_TOPMOST, 0, 0, 0, 0, SWP_NOMOVE|SWP_NOSIZE);
        break;
    case WM_USERMSG:
        if(lpMapped != NULL) {
          RECT r;
          int w, h;
          char buf[1024];
          GetWindowRect(hWnd, &r);
          w = r.right  - r.left;
          h = r.bottom - r.top;
          hdc = GetDC(hWnd);
          SelectObject(hdc, GetStockObject(LTGRAY_BRUSH));
          PatBlt(hdc, 0, 0, w, h, PATCOPY);
          strcpy(buf, "> ");
          strcat(buf, lpMapped);
          SetBkMode(hdc, TRANSPARENT);
          TextOut(hdc, 10, 50, buf, strlen(buf));
          ReleaseDC(hWnd, hdc);
        }
        break;
    case WM_CHAR:
        if ((char)wParam != 'q') break;
    case WM_DESTROY:            /* exit program */
        if(lpMapped != NULL) UnmapViewOfFile(lpMapped);
        if(hMapFile != NULL) CloseHandle(hMapFile);
        PostQuitMessage(0);
        break;
    default:
        return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}
```

索引