Origin C プログラミングガイド

最終更新 2017 年 10 月

Copyright © 2017 by OriginLab Corporation

このマニュアルのいかなる部分も、OriginLab Corporation の文書による許可無く、理由の如何に因らず、どのような形式であっても複製または送信することを禁じます。

OriginLab、Origin、LabTalk は、OriginLab Corporation の登録商標または商標です。その他、記載されている会社名、製品名は、各社の商標および登録商標です。

このマニュアルは、(株)ライトストーンの協力により、翻訳・制作したものです。

OriginLab Corporation
One Roundhouse Plaza
Northampton, MA 01060
USA
(413) 586-2013
(800) 969-7720
Fax (413) 585-0126
www.OriginLab.com

目次

1 Origin C プログラミングガイド 1

2 基本機能 3

2.1 Hello World チュートリアル 3

3 言語の基本 7

3.1 データ型と変数 7
3.2 演算子 10
3.3 フロー制御ステートメント 13
3.4 関数 17
3.5 クラス 19
3.6 エラーと例外処理 20

4 事前定義されたクラス 23

4.1 分析クラス 23
4.2 アプリケーションコミュニケーションクラス 23
4.3 コンポジットデータ型のクラス 23
4.4 内部 Origin オブジェクトクラス 25
4.5 システムクラス 32
4.6 ユーザインターフェースのコントロールクラス 32
4.7 Utility クラス 35

5 Origin C コードの作成と使用 37

5.1 Origin C ファイルの作成と編集 37
5.2 コンパイル、リンク、ロード 42
5.3 デバッグする 46
5.4 コンパイルした関数を使用する 47
5.5 Origin C コードを配布する 51

6 行列ブック、行列シートと行列オブジェクト 55

6.1 行列ブックの基本操作 55
6.2 行列シート 58
6.3 行列オブジェクト 66

7 ワークブック、ワークシート、ワークシート列 77

7.1 ワークブック 77
7.2 ワークシート列 82
7.3 ワークシート 90

8 グラフ 119

8.1 グラフの作成と編集 120
8.2 データプロットを追加する 126
8.3 データプロットを編集する 133
8.4 レイヤを管理する 146
8.5 グラフィックオブジェクトの作成とアクセス 151

9 データ操作 159

9.1 数値データ 159
9.2 文字列データ 166
9.3 日付と時間データ 169
10 プロジェクト 171
10.1 プロジェクト管理 171
10.2 フォルダを管理する 172
10.3 ページにアクセスする 173
10.4 メタデータにアクセスする 175
10.5 操作にアクセスする 180
11 インポート 183
11.1 データをインポートする 183
11.2 画像のインポート 189
11.3 動画のインポート 192
12 エクスポート 195
12.1 ワークシートのエクスポート 195
12.2 グラフのエクスポート 195
12.3 行列のエクスポート 196
12.4 動画のエクスポート Video Writer 197
13 分析とアプリケーション 199
13.1 数学 199
13.2 統計 208
13.3 カーブフィッティング 210
13.4 信号処理 229
13.5 ピークと基線 231
13.6 NAG 関数を使用する 235
14 出力オブジェクト 243
14.1 結果ログ 243
14.2 スクリプトウィンドウ 243
14.3 ノートウィンドウ 244
14.4 レポートシート 244
15 データベースへのアクセス 245
15.1 データベースからインポート 245
15.2 データベースへのエクスポート 246
15.3 SQLite データベースへのアクセス SQLite 248
16 LabTalk へのアクセス 251
16.1 LabTalk 変数の値を取得およびセットする 251
16.2 LabTalk スクリプトを実行する 252
16.3 Origin C コードに LabTalk スクリプトを埋め込む 253
17 X ファンクションへのアクセス 255
17.1 X ファンクション impFile を Origin C から呼び出す 255
18 ユーザインターフェース 257

18.1 ダイアログ 257
18.2 ウェイトカーソル 303
18.3 グラフからデータポイントを取得 304
18.4 グラフにコントロールを追加する 305
19 外部リソースへのアクセス 309
19.1 サードパーティ製 DLL 関数にアクセスする 309
19.2 外部アプリケーションにアクセスする 334
20 リファレンス 337
20.1 クラスの階層 337
20.2 コレクション 340
21 索引 343

1 Origin C プログラミングガイド

Origin には 2 つのプログラミング言語があります。Origin C と LabTalk です。

このガイドは Origin C プログラミング言語について説明しています。また、ダイアログビルダダイアログを作成し制御する方法も示しています。ダイアログビルダは、フローティングツール、ダイアログボックス、ウィザードのようなカスタムダイアログを作成および制御することができます。

読者は、オブジェクト指向プログラミングコンセプトを含む C/C++ 言語に慣れ親しんでいることとしています。

詳細なサンプルを含む最新のサンプルのソースコードは、Origin C Language Reference にあります。

2 基本機能

Origin C は、ANSI C プログラミング言語のシンタックスに基づくハイレベルなプログラミング言語です。Origin C は クラス、ストリーム内での変数宣言、オーバーロード関数、参照、デフォルトの関数引数などを含む多くの C++の機能をサポートしています。Origin C は、C#プログラミング言語からコレクションおよび **foreach** と **using** ステートメントをサポートしています。

Origin C プログラムは、コードビルダという Origin の開発環境(IDE)で開発されます。コードビルダには、シンタックスのハイライト、ワークスペースウィンドウ、コンパイラ、リンカ、デバッガを持つソースコードのエディタを含んでいます。詳細は、**ヘルプ:プログラミング:コードビルダ**を確認してください。

Origin C を使うと、Origin のデータインポートと取り扱い、グラフ作成、分析、イメージエクスポート機能など完全に制御することができます。Origin C で作成したアプリケーションは、Origin のスクリプト言語である LabTalk より高速に実行します。

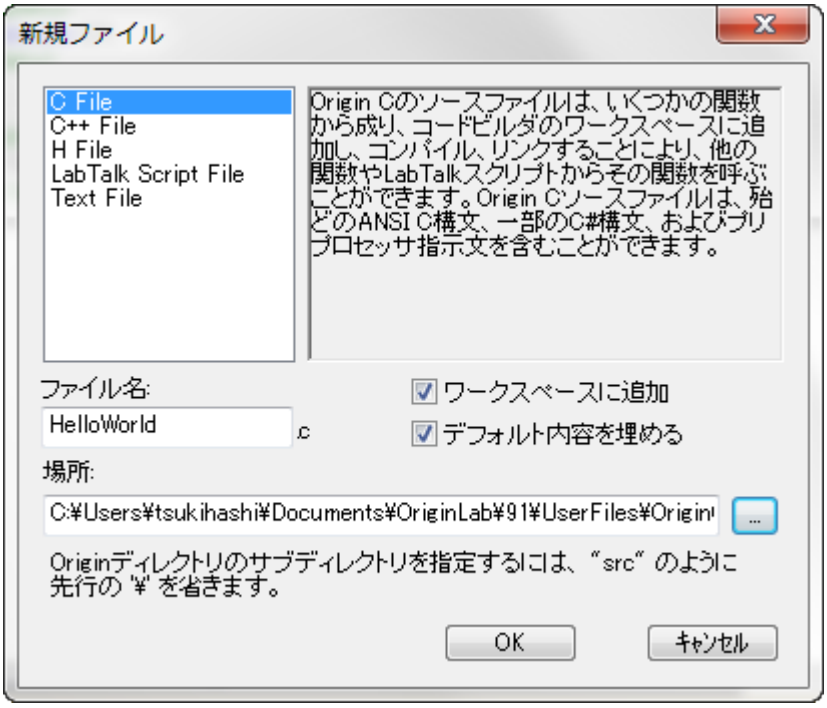
2.1 Hello World チュートリアル

このチュートリアルは、Origin C 関数を作成する**コードビルダ**を使用して、Origin から関数にアクセスする方法を示しています。関数自体はとても単純ですが、ここで提供されているステップは、Origin C 関数の記述を始める手助けとなります。

1. 標準ツールバーの**コードビルダ**ボタン をクリックし、**コードビルダ**を開きます。

2. **コードビルダ**で、コードビルダの**新規作成**ボタン をクリックし、**新規ファイル**ダイアログを開きます。

3. ダイアログのリストボックスから **C File** を選択し、**ファイル名**テキストボックスで *HelloWorld* と入力します。



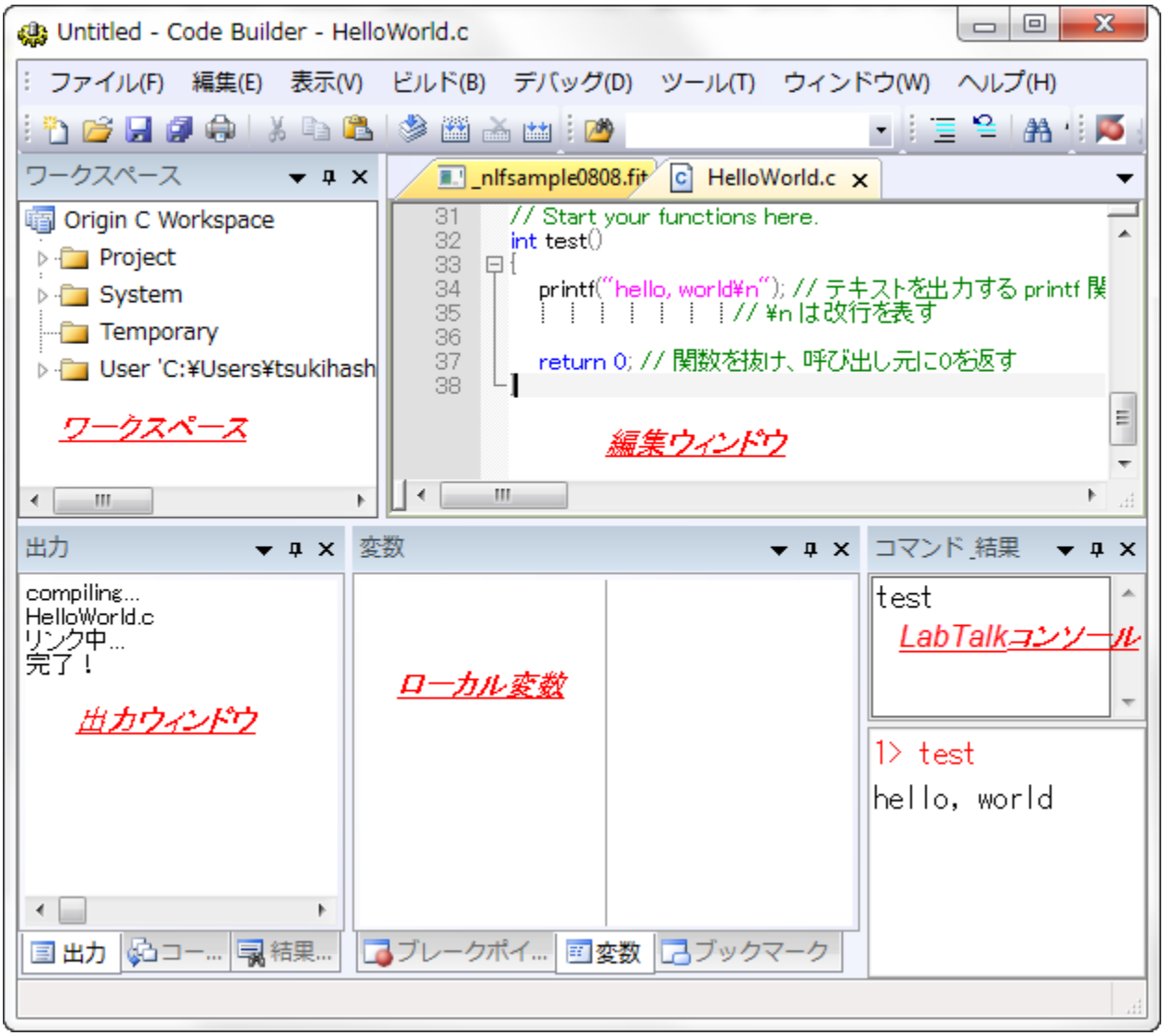
4. **OK** をクリックすると、新しいファイルがコードビルダのマルチドキュメントインターフェース(MDI)で開きます。

5. 次の Origin C コードをコピーし、*// Start your functions here* という行の下に入力します。

```
int test()

{

    printf("hello, world¥n"); // テキストを出力する printf 関数 の呼び出し

                              // ¥n は改行を表す

    return 0; // 関数を抜け、呼び出し元に 0 を返す

}
```

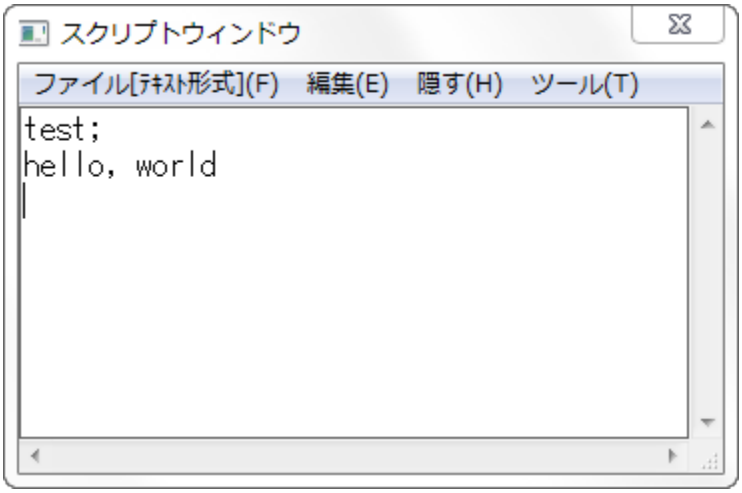
6. **コードビルダ**の標準ツールバーにあるビルド ボタン をクリックして、*HelloWorld.C* ソースファイルをコンパイル、リンクします。

   コードビルダの**出力**ウィンドウは次のように表示されます。



7. Origin でこの関数を使用することができます。例えば、Origin の**スクリプトウィンドウ**でこの関数を呼び出すことができます。**スクリプトウィンドウ**が開いていない場合、メニューから**ウィンドウ**:**スクリプトウィンドウ**と選択して、開きます。

8. **スクリプトウィンドウ**に、*test* という関数名を入力し、ENTER キーを押してコマンドを実行します。Origin C 関数が実行され、*hello, world* という文字が次の行に表示されます。



9. **スクリプトウィンドウ**に加え、コードビルダの **LabTalk コンソールウィンドウ**からも関数を呼び出すことができます。この**コンソールウィンドウ**が開いていない場合、コードビルダのメニューで**表示**:**LabTalk コンソール**を選択します。

Origin C ファイルが問題なくコンパイルリンクされると、ファイル内で定義したすべての関数は、LabTalk スクリプトをサポートしている Origin 内のどこからでもスクリプトコマンドとして呼び出すことができます。関数パラメータおよび戻り値は、スクリプトからアクセス可能なある基準に合致する必要があります。詳細については、LabTalk ヘルプファイルの **LabTalk プログラミング**:**LabTalk ガイド**:**X ファンクションおよび Origin C 関数の呼び出し**: **Origin C 関数** の章をご覧ください。このヘルプファイルは Origin の**ヘルプ**:**プログラミング**:**LabTalk** メニューからアクセスできます。

3 言語の基本

Origin C は、ANSI C/C++プログラミング言語に基づいています。これは、Origin C が同じデータ型、演算子、フロー制御、ユーザ定義の関数、クラス、エラー、例外処理をサポートしているということです。次のセクションでは、Origin C のこれらの領域について詳細に説明します。

3.1 データ型と変数

3.1.1 ANSI C データタイプ

Origin C は、char, short, int, float, double, void 型などのすべての ANSI C データ型をサポートしています。さらに、これらのデータ型のそれぞれを配列にすることができ、ポインタで指定することもできます。

```
char name[50];       // 文字列の配列を宣言

unsigned char age;   // 符号無し 8 ビット整数型を宣言

unsigned short year; // 符号無し 16 ビット整数型を宣言
```

3.1.2 Origin C の合成データ型

配列を宣言する C シンタックスがサポートされていますが、Origin C は 1 次元または 2 次元配列のデータタイプの操作を簡単にする **string**, **vector**, **matrix** クラスを提供しています。これらのデータ型は、char, byte, short, word, int, uint, complex を含みます。ベクター型は、文字列配列のの型にできますが、行列はできません。行列は数値型のみにすることができます。行列は数値型のみにすることができます。

```
string str = "hello, world¥n";        // 文字列を宣言し、初期化

vector<double> vA1 = {1.5, 1.8, 1.1}; // double 型の宣言と初期化

vector vA2 = {2.5, 2.8, 2.1, 2.4};

vector<string> vs(3);                 // 文字列配列を宣言

vs[0] = "This ";                      // 文字列を各文字列配列の項目に割り当て

vs[1] = "is ";

vs[2] = "test";

matrix<int> mA1;                      // 整数の行列を宣言

matrix mA2;                           // double 型の行列を宣言

// NOTE: double 型は、データ型が、ベクターおよび行列変数の

// 宣言で指定されていないときに、暗黙に使われます。
```

Origin C で提供される別の役立つクラスは、**complex** クラスです。complex クラスは、実数と虚数の両方を含む数値データ型をサポートします。

```
complex cc(4.5, 7.8);          // 複素数値を宣言
                               // 実数コンポーネントは 4.5 にセットされ
                               // 虚数のコンポーネントは 7.8 にセット
out_complex("value = ", cc); // 出力は複素数値
```

3.1.3 色

Origin C の色は、DWORD 値で表されます。これらの値は、Origin の内部カラーパレットへのインデックスにしたり、実際に RGB 値を混合することができます。

パレットのインデックス

Origin の内部のパレットは 24 色含んでいます。Origin の内部のカラーパレットへのインデックスは 0 から 23 までの値です。Origin C は、これらの値に固定の名前を提供しています。それぞれの名前は、プレフィックス SYSCOLOR_ に色の名前を続けます。次の表は、24 色の名前とインデックスです。

インデックス	名前	インデックス	名前
0	SYSCOLOR_BLACK	12	SYSCOLOR_DKCYAN
1	SYSCOLOR_RED	13	SYSCOLOR_ROYAL
2	SYSCOLOR_GREEN	14	SYSCOLOR_ORANGE
3	SYSCOLOR_BLUE	15	SYSCOLOR_VIOLET
4	SYSCOLOR_CYAN	16	SYSCOLOR_PINK
5	SYSCOLOR_MAGENTA	17	SYSCOLOR_WHITE
6	SYSCOLOR_YELLOW	18	SYSCOLOR_LTGRAY
7	SYSCOLOR_DKYELLOW	19	SYSCOLOR_GRAY
8	SYSCOLOR_NAVY	20	SYSCOLOR_LTYELLOW
9	SYSCOLOR_PURPLE	21	SYSCOLOR_LTCYAN
10	SYSCOLOR_WINE	22	SYSCOLOR_LTMAGENTA
11	SYSCOLOR_OLIVE	23	SYSCOLOR_DKGRAY

```
DWORD dwColor = SYSCOLOR_ORANGE;
```

自動カラー

自動として参照される特別なカラーインデックスがあります。このインデックスが使われると、要素はその親と同じ色で色付けされます。すべての要素が**自動**インデックスをサポートしているわけではありません。要素に対する Origin のグラフィカルユーザインターフェースを見て、**自動**インデックスがサポートされているかどうかを決定します。

INDEX_COLOR_AUTOMATIC マクロは、**自動**インデックス値が必要なときに使用します。

```
DWORD dwColor = INDEX_COLOR_AUTOMATIC;
```

RGB

Origin の色の値は、RGB 値を表すこともできます。RGB 値は、8 ビットの赤、緑、青の成分で構成されます。これらの値は、RGB マクロを使って簡単に行うことができます。

```
DWORD brown = RGB(139,69,19); // 茶色
```

RGB マクロから返される値は、Origin の色の値として直接使うことができません。RGB2OCOLOR マクロを使って、RGB 値を Origin の色の値に変換する必要があります。

```
DWORD brown = RGB2OCOLOR(RGB(139,69,19)); // 茶色
```

Origin の色値が RGB 値を表しているのか、パレット内のインデックスを表しているのかを知る必要がある場合、OCOLOR_IS_RGB マクロを使うことができます。値が RGB 値を表す場合、このマクロは True を返し、それ以外の場合 False を返します。

```
if( OCOLOR_IS_RGB(ocolor) )

    out_str("color value represents an RGB color");

else

    out_str("color value represents a color index");
```

Origin の色の値が RGB 値を表すことが分かったら、GET_CRF_FROM_RGBOCOLOR マクロを使って、Origin の色の値から RGB 値を抽出することができます。

```
if( OCOLOR_IS_RGB(ocolor) )

{

    DWORD rgb = GET_CRF_FROM_RGBOCOLOR(ocolor);

    printf("red = %d, green = %d, blue = %d¥n",

        GetRValue(rgb), GetGValue(rgb), GetBValue(rgb));

}
```

3.2 演算子

演算子は、ANSI C と同じ算術演算子、論理演算子、比較演算子、ビット演算子をサポートします。次のセクションは 4 種類の演算子とその用法を示します。

3.2.1 算術演算子

演算子	目的
*	乗法
/	除法
%	剰余(余り)
+	加法
-	減法
^	指数 下記 note を参照

Note:デフォルトで、Origin C は、指数演算子としてキャレット(^)を扱います。これは LabTalk と一貫性を持つために行われています。ANSI C では、キャレット(^)を排他論理和演算子として使います。コードの前に特別な pragma ステートメントを使って強制的に Origin C がキャレット(^)を排他論理和演算子(OR)として使うようにすることができます。

```
out_int("10 raised to the 3rd is ", 10^3);

#pragma xor(push, FALSE)

out_int("10 XOR 3 is ", 10^3);

#pragma xor(pop) // xor の動作をデフォルトに戻す
```

整数を別の整数で除算すると整数の結果となります。以下の pragma ステートメントをコード前に使用して、Origin C コンパイラがすべての数値定数を double として扱うようにできます。

```
out_double("3/2 is ", 3/2); // 1 が出力

#pragma numlittype(push, TRUE)

out_double("3/2 is ", 3/2); // 1.5 が出力

#pragma numlittype(pop) // numlittype のデフォルトに戻す
```

剰余演算子は、左オペランドを右オペランドで割った余りを計算します。この演算子は整数のオペランドにのみ適用します。

```
out_int("The remainder of 11 divided by 2 is ", 11 % 2);
```

3.2.2　比較演算子

比較演算子は、True または False を評価し、True は 1、False は 0 を返します。

演算子	目的
>	より大きい
>=	以上
<	より小さい
<=	以下
==	等しい
!=	等しくない

```
if( aa >= 0 )
    out_str("aa is greater than or equal to zero");

if( 12 == aa )
    out_str("aa is equal to twelve");

if( aa < 99 )
    out_str("aa is less than 99");
```

3.2.3　理論演算子

論理演算子は、True または False を評価し、True は 1、False は 0 を返します。オペランドは左から右へ評価されます。論理式全体が決まったら評価が終わります。

演算子	目的
!	等しくない
&&	および
||	または

次の 2 つのサンプルを考えます。

```
expr1A && expr2

expr1B || expr2
```

expr1A が False または expr1B が True として評価されると、expr2 は評価されません。この動作は、プログラマにはメリットであり、効率的なコードを記述できます。次は、順序の重要性を明確に表したものです。

```
if( NULL != ptr && ptr->dataValue < upperLimit )

    process_data(ptr);
```

上記の例で、ptr が NULL に等しいと、if 式全体は False に評価されます。ptr が NULL の場合、NULL ポインタから dataValue メンバーを読み取ると、アプリケーションが直ちに終了するので、dataValue が upperLimit と比較されないことが重要です。

3.2.4 ビット演算子

ビット演算子は個々のビットをテストし、設定します。演算子は、オペランドをビットの配列として扱います。ビット演算子のオペランドは不可欠なタイプでなければなりません。

演算子	目的
~	補数
<<	左にシフト
>>	右にシフト
&	および
^	排他論理和(XOR) 下記 Note を参照
|	論理和 OR

Note:デフォルトで、Origin C は、指数演算子としてキャレット(^)を扱います。これは LabTalk と一貫性を持つために行われています。ANSI C では、キャレット(^)を排他論理和演算子として使います。コードの前に特別な pragma ステートメントを使って強制的に Origin C がキャレット(^)を排他論理和演算子(OR)として使うようにすることができます。

```
out_int("10 raised to the 3rd is ", 10^3);

#pragma xor(push, FALSE)

out_int("10 XOR 3 is ", 10^3);

#pragma xor(pop)
```

3.3 フロー制御ステートメント

Origin C は、if, if-else, switch, for, while, do-while, goto, break, continue を含む ANSI C のすべてのフロー制御をサポートしています。さらに、Origin C は C#の foreach サポートし、オブジェクトのコレクションをループすることができます。

3.3.1 if ステートメント

if ステートメントは、条件分岐に使われ、条件が True の場合にステートメントブロックを実行します。**if-else** ステートメントは、**if** ステートメントに似ていますが、**if-else** ステートメントは、条件の結果が False の場合に代替えのステートメントブロックを実行します。

次は、異なる入力タイプを使った、Origin C の **if** ステートメントのサンプルです。

```
bool bb = true;    // ブール型

if( bb )

{

        out_str("bb is true");

}



int nn = 5;

if( nn )           // 整数型, 0 = false, 非ゼロ = true

{

        out_str("nn not 0");

}



double* pData = NULL;

if( NULL == pData ) // ポインタが NULL の場合チェック

{

        out_str("Pointer pData is NULL");

}
```

次は、Origin C のシンプルな **if-else** ブロックです。if ブロックと else ブロックがブレース{}で囲まれています。

```
if( bRet )

{

        out_str("Valid input");      // bRet が true のとき
```

```
}

else

{

        out_str("INVALID input");    // bRet が false のとき

}
```

if ステートメントが 1 つのステートメントしか持たない場合、ブレースは任意です。これは、上記のコードがブレース無しで書けるということです。

```
if( bRet )

        out_str("Valid input");      // bRet が true のとき

else

        out_str("INVALID input");    // bRet が false のとき
```

3.3.2 switch ステートメント

switch ステートメントは、相互に排他的な選択肢によって、異なるステートメントブロックを実行する必要があるときに使われます。

ケースは、昇順の整数で、*switch* ステートメントに整数の引数で与えられる数値で開始します。*break* コマンドは、どのケースでも switch ブロックを抜けます。

```
switch( nType ) // 条件として整数型の値

{

case 1:

case 2:

    out_str("Case 1 or 2");

    break;



case 3:

    out_str("Case 3");

    // break キーワードがないと case 4 を実行



case 4:

        out_str("Case 4");

        break;
```

```
default:
    out_str("Other cases");
    break;
}
```

3.3.3 for ステートメント

for ステートメントは、指定した回数または各要素がインデックスで参照しているデータの配列によって、1 つ以上のステートメントを実行するのに使われます。

```
char str[] = "This is a string";
for( int index = 0; index < strlen(str); index++ )
{
        printf("char at %2d is %c¥n", index, str[index]);
}
```

3.3.4 while/do-while ステートメント

while と **do-while** ステートメントは、条件が合致するまでステートメントブロックを実行します。**while** ステートメントは、ループの最初に条件を検査し、**do-while** ステートメントは、ループの最後に条件を検査します。

```
int count = 0;
while( count < 10 ) // 条件が True の場合にステートメントを実行
{
    out_int("count = ", count);
    count++;
}

int count = 0;
do
{
        out_int("count = ", count);
        count++;
} while( count < 10 ); // 条件が True の場合にステートメントを実行
```

3.3.5 ジャンプステートメント

ジャンプステートメントは、関数の範囲内で、無条件に別のステートメントに ジャンプするのに使います。**break**, **continue**, **goto** ステートメントがジャンプステートメントです。次のサンプルはこれらのジャンプステートメントを示します。

break

```
for( int index = 0; index < 10; index++ )
{
        if( pow(index, 2) > 10 )
                break; // ループの終了

        out_int("index = ", index);
}
```

continue

```
printf("The odd numbers from 1 to 10 are:");
for( int index = 1; index <= 10; index++ )
{
        if( mod(index, 2) == 0 )
                continue; // 次のインデックス

        printf("%d¥n", index);
}
```

goto

```
    out_str("Begin");
    goto Mark1;

    out_str("Skipped statement");

Mark1:
    out_str("First statement after Mark1");
```

3.3.6 foreach ステートメント

foreach ステートメントは、オブジェクトのコレクションをループするのに使われます。次のコードは、プロジェクト内のすべてのページをループし、その名前とタイプを出力します。

```
foreach(PageBase pg in Project.Pages)
{
    printf("%s is of type %d¥n", pg.GetName(), pg.GetType());
}
```

Origin C のクラスに基づくすべての Collection のリストについては、コレクション のセクションを参照してください。

3.4 関数

3.4.1 グローバル関数

Origin C はさまざまな操作を実行する数多くのグローバル関数を提供しています。これらのグローバル関数は 26 個のカテゴリーに分類されます。

1. 基本 IO
2. 文字および文字列操作
3. COM
4. 通信
5. 曲線
6. データ変換
7. データ範囲
8. 日時
9. ファイル IO
10. ファイル管理
11. フィッティング
12. 画像処理
13. インポートとエクスポート
14. 内部 Origin オブジェクト
15. LabTalk インターフェース
16. 数学関数
17. 数学
18. 行列変換とグリッディング

19. メモリ管理
20. NAG
21. 信号処理
22. スペクトル分析
23. 統計
24. システム
25. ツリー
26. ユーザインターフェース

サンプル付きの完全な関数リストについては、グローバル関数 セクションをご覧ください。

3.4.2 ユーザ定義関数

Origin C は、ユーザ定義関数をサポートしています。Origin C のプログラマは引数の選択肢および戻り型を受け付ける関数を作成することができます。それらの関数は、その引数を操作し、目的を成し遂げます。

以下のコードは、1 つの引数だけで **double** 型の値を返し、**double** 型の値を受け入れる *my_function* という関数を作成します。

```
double my_function(double dData)

{

        dData += 10;

        return dData;

}
```

以下の断片的なコードは、上記の関数を呼び出す方法を示しています。

```
double d = 3.3;          // double 型の値として d を宣言

d = my_function(d);      // 上記関数の呼び出し

out_double("d == ", d); // 'd' の新しい値を出力
```

Origin C 関数は、LabTalk から呼び出すことができます。

```
d = 3.3;             // 'd'に 3.3 を割り当て

d = my_function(d); // 上記関数の呼び出し

d=;                  //  'd' の新しい値を出力
```

3.5 クラス

Origin C は、数多くの組み込みクラスをサポートしていますが、ユーザ自身で作成することもできます。

3.5.1 Origin 定義クラス

Origin C には、Origin の異なるデータ型とユーザインターフェースオブジェクト操作する事前定義のクラスがあります。これらのクラスを使うと、操作を実行する Origin C コードを素早く書くことができます。このセクションは、基本クラスについて、これらのクラスが提供している機能の概要を説明しています。Origin の定義クラスの詳細およびサンプルについては、次の章、事前定義のクラス または Origin C の wiki サイトをご覧ください。

3.5.2 ユーザ定義クラス

Origin C は、ユーザ定義クラスをサポートしています。ユーザ定義クラスでは、Origin C のプログラマがメソッド(メンバー関数)とデータメンバーを持つ自分自身のオブジェクトを作成することができます。

次のコードは、2 つのメソッド **GetName** と **SetName** を持つ **Book** というクラスを作成するものです。

```
class Book

{

public:

        string GetName()

        {

                return m_strName;

        }



        void SetName(LPCSTR lpcszName)

        {

                m_strName = lpcszName;

        }

private:

        string m_strName;

};
```

そして、以下は、上記のメソッドとクラスの定義を使った簡単なサンプルで、**Book** クラスのインスタンスを宣言し、**SetName** を使って名前を付け、**GetName** を使って名前を出力します。

```
void test_class()

{

        Book OneBook; // Book オブジェクトを宣言

        // Book オブジェクトに対して名前をセット/取得する関数の呼び出し

        OneBook.SetName("ABC");

        out_str(OneBook.GetName());

}
```

上記のサンプルはとても単純なものです。クラスの機能、例えば、コンストラクタ/デコンストラクタやバーチャルメソッドなどについて詳しく知りたければ、*Origin の\Samples\Origin C Examples\Programming Guide\Extending Origin C* サブフォルダにある EasyLR.c、EasyLR.h、EasyFit.h ファイルをご覧ください。

3.6 エラーと例外処理

Origin C は、C++ の **try**, **catch**, **throw** ステートメントを使って、例外処理をサポートしています。

try ブロックは、**try** キーワードの後にブレースで囲んだ 1 つ以上のステートメントで構成されます。try ブロックの直後は、catch ハンドラーです。Origin C は、*整数*の引数を受け付ける catch ハンドラーを 1 つだけサポートします。**catch** キーワードの後には、ブレースで囲まれた 1 つ以上のステートメントとなります。

```
try

{

    LPSTR lpdest = NULL;    // NULL ポインタ

    strcpy(lpdest, "Test"); // エラーを起こすために NULL ポインタにコピー

}

catch(int nErr)

{

    out_int("Error = ", nErr);

}
```

try ブロック内のステートメントを実行することで try-catch が動作します。エラーが発生すると、実行は catch ブロックにジャンプします。エラーが発生しなければ、catch ブロックは無視されます。

throw キーワードは任意で、エラーをトリガーにして、catch ブロックへのジャンプを実行します。

```
void TryCatchThrowEx()
{
    try
    {
        DoSomeWork(4);  // 成功を表示するために有効な数字を渡す
        DoSomeWork(-1); // エラーを引き起こすために無効な数字を渡す
    }
    catch(int iErr)
    {
        printf("Error code = %d¥n", iErr);
    }
}
void DoSomeWork(double num)
{
    if( num < 0 )
        throw 100; // 強制的にエラー
    if( 0 == num )
        throw 101; // 強制的にエラー

    double result = sqrt(num) / log(num);
    printf("sqrt(%f) / log(%f) = %g¥n", num, num, result);
}
```

4 事前定義されたクラス

このセクションでは、Origin C の事前定義のクラスについて説明しています。Origin C 組込クラス間の関係についての詳細な情報は、クラス階層 をご覧ください。

4.1 分析クラス

次のクラスは、データ分析を実行するのに使われます。詳細については、次の Origin ヘルプファイルを参照してください。**Origin C:Origin C Reference:Classes:Analysis**

クラス	簡単な説明
NLFitContext	このクラスは、フィット関数の情報だけでなく、Origin C でフィット関数を組み込むことで生成される現在の評価の状態にアクセスするメソッドを提供します。
NLFitSession	このクラスはより高いレベルの Origin クラスです。フィットの評価手順を組み込む目的で、わかりやすいインターフェースを持つ NLFit を包含します。それは、NLFit ダイアログのカーネルです。このクラスは、Origin のインターフェースの処理から生じる複雑さを処理できるので、Origin C でコーディングすることをお勧めします。

4.2 アプリケーションコミュニケーションクラス

次のクラスは、Origin と他のアプリケーション間との通信を可能にするのに使用されます。詳細については、次の Origin ヘルプファイルを参照してください。**Origin C:Origin C Reference:Classes:Application Communication**

クラス	簡単な説明
Matlab	Origin と MATLAB 間の通信を可能にするのに使用されます。

4.3 コンポジットデータ型のクラス

以下のクラスは、コンポジットデータ型のクラスです。詳細については、次の Origin ヘルプファイルを参照してください。**Origin C:Origin C Reference:Classes:Composite Data Types** の章を確認してください。

クラス	簡単な説明
CategoricalData	CategoricalData 型のデータセットは、整数の配列です。この配列は、テキスト 型の Origin の内部データセットに結びつけられ、動的に割り当てられます。この型のデータセットは(1 から始まる)インデックスを参照してカテゴリーにテキスト値をマッピングします 。マッピングインデックスのテキスト値は、CategoricalMap のデータメンバーに保存されます。
CategoricalMap	CategoricalMap 型のデータセットはテキスト値の配列です。この配列は、動的に割り当てられ、サイズが決まりますが、Origin の内部データセットに結びつけられません。このデータセットは、固有のテキスト値を含み、これは英数字でソートされ、通常 CategoricalData タイプに結びついたオブジェクトの要素で参照されます。

complex	このクラスは、複素数型の数値データを扱うのに使われます。複素数の実数部と虚数部の両方を含みます。
Curve	このクラスは、curvebase および vectorbase クラスから派生し、メソッドとプロパティが継承されています。Curve 型のオブジェクトは、GraphLayer クラスで定義されたメソッドを使って簡単にプロットでき、Y データセットと関連の X データセットで構成されます。例えば、行番号に対してプロットされるデータセットは、関連の X データセットを含みません。
curvebase	vectorbase クラスから派生したこのクラスは、メソッドとプロパティを継承し、基底クラスであり、Curve 型のクラスを取り扱うのに使用します。curvebase 型のオブジェクトは構築することができず、Curve のような派生クラスが代わりに使われます。
Dataset	このクラスは、vector および vectorbase クラスから派生し、そのメソッドとプロパティを継承しています。Dataset は配列であり、動的に割り当てられ、大きさが決まります。Origin の内部データセットに結びつけることも、結びつけないこともできます。デフォルトで、Dataset は double 型ですが、char, byte, short, word, int, uint, complex などの基本データ型にすることもできます (string は不可)。シンタックス Dataset<*type*> は、Dataset のこれらの型を構築するのに使うことができます。
Matrix	このクラスは、matrix および matrixbase クラスから派生し、そのメソッドとプロパティを受け継いでいます。Matrix (M は大文字)は 2 次元配列で、動的に割り当てられ、大きさが決まり、Origin の内部行列ウィンドウに結びついています。Matrix のデフォルトの型は、double 型ですが、char, byte, short, word, int, uint, complex などの基本データ型にすることもできます (string は不可)。シンタックス Matrix<*type*> は、Matrix のこれらの型を構築するのに使うことができます。 このクラスは、内部の Origin 行列内のデータにアクセスするのに使われ、MatrixObject クラスは行列のスタイルを制御するのに使われます。すなわち、MatrixObject と Matrix クラスの関係は、Column と Dataset クラスの関係と同じです。 Origin 行列(Matrix オブジェクトで参照)内のセルに表示されているデータ値は、通常、ワークシート内では、Z 値として参照され、関連する X と Y 値は、それぞれ行列の列と行に線形でマッピングされます。
matrix	このクラスは、matrixbase クラスから派生し、そのメソッドとプロパティを受け継いでいます。matrix (m は小文字)は 2 次元配列で、動的に割り当てられ、大きさが決まり、Origin の内部行列ウィンドウには結びついておらず、柔軟性が提供されます。matrix のデフォルトの型は、double 型ですが、char, byte, short, word, int, uint, complex などの基本データ型にすることもできます (string は不可)。シンタックスmatrix<*type*> は、matrix のこれらの型を構築するのに使うことができます。
matrixbase	このクラスは、matrix と Matrix クラスを取り扱う基底クラスです。つまり、matrixbase 型のオブジェクトは構築することができず、matrix や Matrix のような派生クラスのオブジェクトが代わりに使われます。
PropertyNode	このクラスは、Bool, int, float, double, string, vector, matrix, picture などの異なるデータ型のプロパティを含めるためだけに使用されます。
string	このクラスは、文字の null で終わる配列を構築するのに使われ、MFC CString オブジェクト

	に似ています。文字列(テキストデータ)を操作する多くのメソッドは、このクラスで定義されます。シンタックス vector<string> で vector クラスと一緒に使って、文字列配列を定義できます。
Tree	このクラスは、Origin C ツリーを XML ファイルとして保存したり、XML ファイルから Origin C ツリーにロードするのに使用できます。
TreeNode	このクラスは、複数レベルのツリーを構築し、ツリーを横断し、ツリーノードの属性にアクセスするいくつかのメソッドを提供します。
TreeNodeCollection	このクラスは、番号付きの名前のプレフィックスで子ツリーノードのコレクションを取得するのに使われます。
vectorbase	このクラスは、vector と Dataset クラスを取り扱う基底クラスです。つまり、この型のオブジェクトは構築することができず、vector や Dataset のような派生クラスのオブジェクトが代わりに使われます。
vector	このクラスは、vectorbase クラスから派生し、そのメソッドとプロパティを受け継いでいます。vector は配列で、動的に割り当てられ、大きさが決まり、Origin の内部データセットには結びついておらず、柔軟性が提供されます。vector のデフォルトの型は、double 型ですが、char, byte, short, word, int, uint, complex, string などの基本データ型にすることもできます。シンタックス vector<*type*> は、vector のこれらの型を構築するのに使うことができます。

4.4 内部 Origin オブジェクトクラス

次のクラスは、Origin オブジェクトを取り扱うのに使われます。詳細については、次の Origin ヘルプファイルを参照してください。**Origin C:Origin C Reference:Classes:Internal Origin Objects** の章を確認してください。

クラス	簡単な説明
Axis	このクラスは、OriginObject クラスから派生され、Origin の軸にアクセスするのに使うことができます。Origin の軸は、Origin ページのレイヤに含まれます。
AxisObject	このクラスは、OriginObject クラスから派生され、軸刻み、グリッド線、ラベルを含む Origin の軸オブジェクトにアクセスするのに使うことができます。Origin の軸オブジェクトは、Origin のグラフページの軸に含まれます。
Collection	このクラスは、ページ(プロジェクトファイル内のすべての PageBase オブジェクトのコレクション)などさまざまな Origin 内部オブジェクトのコレクションのテンプレートを提供します。このクラスは暗黙のテンプレート化された型 _TemplType を含み、これはコレクションの 1 つの要素の型です。例えば、Project クラスの Pages コレクションのテンプレート化された型 (Collection<PageBase> Pages;) は PageBase です。 各コレクションは、通常親クラスを持ち、そのデータメンバーはコレクションです。例えば、Project には、すべてのページが含まれるので、Collection<PageBase> Pages は Project クラスのメンバーの 1 つです。そのため、各コレクションは内部オブジェクトに接続されたり、接続されなかったりできます。

	すべてのコレクションは、Collection クラスで定義されるメソッドを使うことができます。foreach ループは、コレクション内の各エレメントを一度にループする最も役立つ方法です。
CollectionEmbeddedPages	このクラスは、ワークシート内に埋め込まれたページにアクセスするのに使われます。
Column	このクラスは、DataObject, DataObjectBase および OriginObject クラスから派生し、そのメソッドとプロパティを継承しています。このクラスで、メソッドとプロパティは Origin ワークシート列に対して提供されます。ワークシートオブジェクトは、Column オブジェクトのコレクションを含み、各 Column オブジェクトは Dataset を持ちます。Column オブジェクトは、主に関連 Dataset のデータのスタイルを制御するのに使用します。 Column オブジェクトは、ラッパーオブジェクトで、これは Origin の内部の列オブジェクトを参照しますが、実際にはそれは Origin 内には存在しません。
DataObject	このクラスは DataObjectBase クラスから派生され、ワークシートの列と行列のオブジェクトの基底クラスです。Origin のデータオブジェクトは、Origin ページのレイヤに含まれます。例えば、列(データオブジェクト)は、ワークシートウィンドウ(ページ)のワークシート(レイヤ)に含まれます。
DataObjectBase	このクラスは抜粋した基底クラスで、DataObject と DataPlot に関連したクラスタイプを取り扱うメソッドとプロパティを提供します。つまり、この型のオブジェクトは構築することができず、DataObject, Column, MatrixObject, DataPlot のような派生クラスのオブジェクトが代わりに使われます。
DataPlot	このクラスは、DataObjectBase および OriginObject クラスから派生し、そのメソッドとプロパティを受け継いでいます。このクラスで、メソッドとプロパティは Origin データプロットに対して提供されます。Origin の内部データプロットオブジェクトは、Origin のデータプロットの特徴を保存するのに使われ、グラフページ上のグラフレイヤに含まれます。 DataPlot オブジェクトはラッパーオブジェクトで、これは Origin の内部のデータプロットを参照しますが、実際にはそれは Origin 内には存在しません。つまり、複数のラッパーオブジェクトは、同じ内部 Origin オブジェクトを参照できます。
DataRange	データ範囲の構築やワークシート、行列、グラフウィンドウにデータをアクセスするために、このクラスにはメソッドとプロパティがあります。このクラスは、データ自体を持たず、ページ名、シート名(グラフに対してレイヤインデックス)、行/列のインデックス(グラフに対してデータプロットのインデックス)でデータ範囲を保持するだけです。 複数データ範囲が DataRange オブジェクトに含まれ、部分データ範囲はデータシート全体、1 つの列、1 つの行、連続した複数列、連続した複数行にすることができます。
DataRangeEx	このクラスは、DataRange の拡張クラスです。
DatasetObject	このクラスは、非数値データセットにアクセスするのに使われ、通常、これは Column オブジェクトのメンバーです。
Datasheet	このクラスは、Layer および OriginObject クラスから派生し、そのメソッドとプロパティを継承しています。このクラスは、Origin のワークシートと行列レイヤを扱うのに使われます。

Folder	プロジェクトエクスプローラは、Windows エクスプローラのようなフォルダ/サブフォルダ構造を持つ Origin 内のユーザインターフェースです。これは、Origin プロジェクトファイル内のグラフ、レイアウト、行列、ノート、ワークシートウィンドウを整理したり、アクセスするのに使用します。 Folder クラスは、プロジェクトエクスプローラのメソッドとプロパティにアクセスでき、すべての Origin ページとプロジェクトエクスプローラフォルダのコレクションを含みます。 Folder オブジェクトは、ラッパーオブジェクトで、これは Origin の内部のプロジェクトエクスプローラオブジェクトを参照しますが、実際にはそれは Origin 内には存在しません。つまり、複数のラッパーオブジェクトは、同じ内部 Origin オブジェクトを参照できます。
fpoint3d	このクラスは、3 次元空間の (x, y, z) 座標に対して double 型のデータポイントを扱うのに使われます。
fpoint	このクラスは、2 次元 (x, y) 座標に対して double 型のデータポイントを扱うのに使われます。
GetGraphPoints	このクラスは、画面上のポイントの位置(x, y)を取得したり、Origin のグラフウィンドウからデータポイントを取得するのに使います。
GraphLayer	このクラスは、Layer および OriginObject クラスから派生し、そのメソッドとプロパティを継承しています。このクラスで、メソッドとプロパティは Origin グラフレイヤに対して提供されます。 内部の Origin グラフページには、1 つ以上のグラフレイヤが含まれ、グラフレイヤには、1 つ以上のデータプロットが含まれます。つまり、GraphPage クラスは GraphLayer オブジェクトのコレクションを含み、GraphLayer クラスは DataPlot オブジェクトのコレクションを含みます。GraphLayer オブジェクトは、ラッパーオブジェクトで、これは Origin の内部のグラフレイヤオブジェクトを参照しますが、実際にはそれは Origin 内には存在しません。複数のラッパーオブジェクトは、同じ内部 Origin オブジェクトを参照できます。
GraphObject	このクラスは、OriginObject クラスから派生し、そのメソッドとプロパティを受け継いでいます。このクラスでは、メソッドとプロパティが Origin のグラフオブジェクトを取り扱うために提供され、これにはテキスト注釈、図形オブジェクト(矩形、矢印、線オブジェクトなど)、データプロットのスタイルフォルダ、関心のある領域(ROI)を含みます。 Origin のグラフオブジェクトは、一般に、Origin ページ上のレイヤに含まれ、GraphLayer クラスは GraphObjects のコレクションを含みます。Graph オブジェクトは、ラッパーオブジェクトで、これは Origin の内部のグラフオブジェクトを参照しますが、実際にはそれは Origin 内には存在しません。複数のラッパーオブジェクトは、同じ内部 Origin オブジェクトを参照できます。
GraphPage	このクラスは、Page, PageBase および OriginObject クラスから派生し、そのメソッドとプロパティを継承しています。このクラスで、メソッドとプロパティは Origin グラフページ(ウィンドウ)を取り扱うために提供されます。GraphPage オブジェクトは、ラッパーオブジェクトで、これは Origin の内部のグラフページオブジェクトを参照しますが、それは Origin 内には存在しません。つまり、複数のラッパーオブジェクトは、同じ内部 Origin オブジェクトを参照できます。 Project クラスは、開いているプロジェクトファイル内の GraphPages という GraphPage オブジェクトのコレクションを含みます。GraphPage オブジェクトは、Origin のグラフページ上

	のレイヤを位置付け、アクセスするのに使われ、DataPlots や GraphicObjects などのレイヤ内のオブジェクトにアクセスするのに使うこともできます。
GraphPageBase	このクラスは、GraphPage と LayoutPage の基底クラスです。
Grid	このクラスは、データシートウィンドウ(Origin のワークシートと行列シート)のフォーマットをセットするのに使われます。データ選択、列/行ラベルの表示、セルのテキストの色、セルの統合など、特別な関数もこのクラスで提供されています。
GroupPlot	このクラスは、OriginObject クラスから派生され、Origin のグループプロットを取り扱うのに使うことができます。GroupPlot のオブジェクトは、Origin ページのレイヤに含まれます。
Layer	このクラスは、OriginObject クラスから派生し、そのメソッドとプロパティを受け継いでいます。このクラスで、メソッドとプロパティは Origin 内部のレイヤを取り扱うために提供されます。ノートページを除く、すべての Origin ページ(ウィンドウ)には 1 つ以上のレイヤを含みます。Origin オブジェクトは、通常ページ上にあり、ページに含まれるレイヤに含まれます。多くのグラフオブジェクトはレイヤに含まれ、Layer クラスはグラフオブジェクトのコレクションを含みます。 Layer オブジェクトは、ラッパーオブジェクトで、これは Origin の内部のレイヤオブジェクトを参照しますが、実際にはそれは Origin 内には存在しません。複数のラッパーオブジェクトは、同じ内部 Origin オブジェクトを参照できます。
LayoutPage	このクラスは、Page, PageBase および OriginObject クラスから派生し、そのメソッドとプロパティを受け継いでいます。このクラスで、メソッドとプロパティは Origin の内部レイアウトページ(ウィンドウ)を取り扱うために提供されます。Project クラスは LayoutPage オブジェクトのコレクションを含みます。 LayoutPage オブジェクトは、ラッパーオブジェクトで、これは Origin の内部のレイアウトページオブジェクトを参照しますが、それは Origin 内には存在しません。複数のラッパーオブジェクトは、同じ内部 Origin オブジェクトを参照できます。
Layout	このクラスは、Layer および OriginObject クラスから派生し、そのメソッドとプロパティを継承しています。このクラスで、メソッドとプロパティは Origin 内部のレイアウトレイヤを取り扱うために提供されます。Origin のレイアウトページは、他のオブジェクトを含むレイアウトレイヤを含みます。 Layout オブジェクトは、ラッパーオブジェクトで、これは Origin の内部のレイアウトオブジェクトを参照しますが、それは Origin 内には存在しません。複数のラッパーオブジェクトは、同じ内部 Origin オブジェクトを参照できます。
MatrixLayer	このクラスは、Datasheet, Layer および OriginObject クラスから派生し、そのメソッドとプロパティを受け継いでいます。このクラスで、メソッドとプロパティは Origin の行列ページの行列レイヤを取り扱うために提供されます。Origin の行列は、多くの行列オブジェクトを含み、MatrixLayer クラスは行列レイヤ内の行列オブジェクトのコレクションを含みます。 MatrixLayer オブジェクトは、ラッパーオブジェクトで、これは Origin の内部の行列レイヤオブジェクトを参照しますが、実際にはそれは Origin 内には存在しません。複数のラッパーオブジェクトは、同じ内部 Origin オブジェクトを参照できます。

MatrixObject	このクラスは、DataObject, DataObjectBase および OriginObject クラスから派生し、そのメソッドとプロパティを継承しています。このクラスは、Origin の行列オブジェクトを扱うのに使われます。 MatrixObject は、内部の Origin 行列内のデータのスタイルを制御するのに使われ、Matrix クラスは行列のデータにアクセスするのに使われます。つまり、Column クラスが Dataset クラスと同じ関係を持つように、MatrixObject クラスは Matrix と同じ関係を持ちます。言い換えれば、ワークシート列(Column)がデータセット(Dataset)を保持するように、Origin の内部行列オブジェクト(MatrixObject)は行列データセット(Matrix)を保持します。行列のセルに表示されているデータ値は、Z 値として参照され、関連する X と Y 値は、それぞれ行列の列と行に線形でマッピングされます。仮に MatrixLayer に対して 1 つの MatrixObject しかないとしても、MatrixLayer は、MatrixObjects のコレクションを持ちます。 MatrixObject は、ラッパーオブジェクトで、これは Origin の内部の行列オブジェクトを参照しますが、実際にはそれは Origin 内には存在しません。複数のラッパーオブジェクトは、同じ内部 Origin オブジェクトを参照できます。
MatrixPage	このクラスは、Page, PageBase および MatrixPage クラスから派生し、そのメソッドとプロパティを受け継いでいます。このクラスで、メソッドとプロパティは Origin の内部行列ページ(ウィンドウ)を取り扱うために提供されます。 MatrixPage オブジェクトは、ラッパーオブジェクトで、これは Origin の内部の行列ページオブジェクトを参照しますが、それは Origin 内には存在しません。複数のラッパーオブジェクトは、同じ内部 Origin オブジェクトを参照できます。 Project クラスは、開いているプロジェクトファイル内の MatrixPages という MatrixPage オブジェクトのコレクションを含みます。MatrixPage オブジェクトは、Origin の行列ページ上のレイヤを位置付け、アクセスするのに使われ、MatrixObjects や GraphicObjects などのレイヤ内のオブジェクトにアクセスするのに使うこともできます。
Note	このクラスは、PageBase および OriginObject クラスから派生し、そのメソッドとプロパティを受け継いでいます。このクラスで、メソッドとプロパティは Origin の内部ノートページ(ウィンドウ)を取り扱うために提供されます。Project クラスは Note オブジェクトのコレクションを含みます。 Note オブジェクトは、ラッパーオブジェクトで、これは Origin の内部のノートページを参照しますが、実際にはそれは Origin 内には存在しません。複数のラッパーオブジェクトは、同じ内部 Origin オブジェクトを参照できます。
OriginObject	このクラスは、すべての Origin オブジェクトの Origin C の基底クラスです。メンバー関数とデータメンバーは、すべての Origin オブジェクトに対して、このクラスで提供されます。
Page	このクラスは、PageBase および OriginObject クラスから派生し、そのメソッドとプロパティを継承しています。このクラスで、1 つ以上のレイヤを含む Origin 内部のページ(ノートウィンドウを除く)を扱うためにメソッドとプロパティが提供されます。Page クラスは、ページ内のレイヤのコレクションを含みます。 Page オブジェクトは、ラッパーオブジェクトで、これは Origin の内部のページオブジェクトを参照しますが、それは Origin 内には存在しません。複数のラッパーオブジェクトは、同じ内部 Origin オブジェクトを参照できます。
PageBase	このクラスは、Origin 内部のページ(ウィンドウ)に対してメソッドとプロパティを提供します。通常、このクラスは 2 つの方法のうちの 1 つで使われます。1 つの方法は、PageBase オブ

	ジェクトを一般的な関数のパラメータとして使い、特定の Page オブジェクトを使わずに行います。もう 1 つの方法は、PageBase オブジェクトを不明なアクティブページに接続するものです。どちらの方法も、特定のページオブジェクトを取り扱うことができます。Note, GraphPage, WorksheetPage, LayoutPage, MatrixPage を含む派生ページ型の抽象クラスとして動作することが、このクラスの目的にもなっています。
point	このクラスは、整数の (x, y) 座標を持つ 2 次元の平面にあるデータポイントを扱うのに使われます。
Project	このクラスは、Origin プロジェクトファイルのほとんどのオブジェクトにアクセスするメソッドとプロパティを提供します。Project クラスは、Project ファイル内の異なるページタイプのコレクション、すべてのデータセット(ワークシート列ではない一時データセット)のコレクションを含みます。このクラスは、ActiveCurve, ActiveLayer, ActiveFolder を含む RootFolder プロパティだけでなく、プロジェクトファイルのアクティブオブジェクトを取得するメソッドを提供します。 Project オブジェクトは、ラッパーオブジェクトで、これは Origin の内部のプロジェクトオブジェクトを参照しますが、実際にはそれは Origin 内には存在しません。一度に 1 つのプロジェクトファイルだけを Origin で開くことができるので、すべての Project オブジェクトは現在開いているプロジェクトファイルを参照します。
ROIObject	このクラスは、GraphObject クラスから派生し、そのメソッドとプロパティを受け継いでいます。このクラスで、メソッドとプロパティは Origin の関心のある領域(ROI)に対して提供されます。Origin の ROI は、Origin 行列内の関心のある領域を識別するのに使われます。 ROIObject は、ラッパーオブジェクトで、これは Origin の内部の ROI オブジェクトを参照しますが、実際にはそれは Origin 内には存在しません。複数のラッパーオブジェクトは、同じ内部 Origin オブジェクトを参照できます。
Scale	このクラスは、OriginObject クラスから派生し、そのメソッドとプロパティを受け継いでいます。このクラスで、メソッドとプロパティは Origin の軸スケールを取り扱うために提供されます。2 つのスケールオブジェクト (X スケールト Y スケール)がグラフページの各グラフレイヤに含まれます。 Scale オブジェクトは、ラッパーオブジェクトで、これは Origin の内部のスケールオブジェクトを参照しますが、実際にはそれは Origin 内には存在しません。これは、複数のラッパーオブジェクトが同じ内部 Origin オブジェクトを参照できるということです。
storage	Origin は、バイナリタイプ(TreeNode 型)と INI タイプ (INIFile 型)の情報を、WorksheetPage, Column, Folder, GraphPage, GraphLayer, DataPlot, Project などの OriginObject から派生した Origin C オブジェクトにすることができる Origin オブジェクトに保存します。
StyleHolder	このクラスは、GraphObject および OriginObject クラスから派生し、そのメソッドとプロパティを継承しています。このクラスで、メソッドとプロパティはデータプロットスタイルホルダに提供されます。データプロットスタイルホルダは、プロットタイプ情報を保存するのに使われます。 Styleholder オブジェクトは、ラッパーオブジェクトで、これは Origin の内部のスタイルホルダオブジェクトを参照しますが、実際にはそれは Origin 内には存在しません。複数のラッパーオブジェクトは、同じ内部 Origin オブジェクトを参照できます。

UndoBlock	このクラスは、プロジェクトに安全にアクセスする 2 つの関数 UndoBlockBegin() と UndoBlockEnd()を提供しています。
WorksheetPage	このクラスは、Page, PageBase および OriginObject クラスから派生し、そのメソッドとプロパティを継承しています。このクラスで、メソッドとプロパティは Origin の内部ワークシートページ(ウィンドウ)に提供されます。Project クラスは WorksheetPage オブジェクトのコレクションを含みます。 WorksheetPage オブジェクトは、ラッパーオブジェクトで、これは Origin の内部のワークシートページオブジェクトを参照しますが、実際にはそれは Origin 内には存在しません。複数のラッパーオブジェクトは、同じ内部 Origin オブジェクトを参照できます。
Worksheet	このクラスは、Datasheet, Layer および OriginObject クラスから派生し、そのメソッドとプロパティを受け継いでいます。このクラスで、メソッドとプロパティは Origin のワークシートページのワークシートレイヤを取り扱うために提供されます。Origin のワークシートは、多くのワークシート列を含み、Worksheet クラスはワークシート内のすべての列のコレクションを含みます。 Worksheet オブジェクトは、ラッパーオブジェクトで、これは Origin の内部のワークシートオブジェクトを参照しますが、実際にはそれは Origin 内には存在しません。複数のラッパーオブジェクトは、同じ内部 Origin オブジェクトを参照できます。
XYRange	このクラスは、DataRange クラスから派生し、そのメソッドとプロパティを受け継いでいます。このクラスで定義したクラスを使うことで、データ範囲は、1 つの独立変数(X)と 1 つの従属変数(Y)を持ち、行列およびワークシートから得ることができ、行列およびワークシートに配置することができます。これはグラフウィンドウにプロットを作成するのに使用することもできます。 DataRange クラスのように、XYRange クラスは、データ自体を持たず、ページ名、シート名(グラフに対してレイヤインデックス)、行/列のインデックス(グラフに対してデータプロットのインデックス)でデータ範囲を保持するだけです。すべての XYRange オブジェクトには複数の XY データ範囲を含めることができます。
XYRangeComplex	このクラスは、XYRange および DataRange クラスから派生し、そのメソッドとプロパティを継承しています。このクラスは、行列とワークシートウィンドウに対する複素数型の XY データセットを取得し、セットするのに使われます。 DataRange クラスのように、XYRangeComplex クラスは、データ自体を持たず、ページ名、シート名、行/列のインデックスでデータ範囲を保持するだけです。すべての XYRangeComplex オブジェクトには複数の XY 複素数データ範囲を含めることができます。
XYZRange	このクラスは、DataRange クラスから派生し、そのメソッドとプロパティを受け継いでいます。このクラスは、行列とワークシートウィンドウに対する XYZ データセットを取得し、セットするのに使われます。 DataRange クラスのように、XYZRange クラスは、データ自体を持たず、ページ名、シート名、行/列のインデックスでデータ範囲を保持するだけです。すべてのすべての XYZRange オブジェクトには複数の XYZ データ範囲を含めることができます。

4.5 システムクラス

次のクラスは、システム設定に関するものです。詳細については、次の Origin ヘルプファイルを参照してください。**Origin C:Origin C Reference:Classes:System**

クラス	簡単な説明
file	このクラスは、バッファされない I/O(ディスクに直接アクセス)を使って、バイナリファイルの読み書きの許可を制御するのに使用されます。これは MFC の CFile クラスに似ています。テキストファイルへのバッファされるストリーム I/O に対する stdioFile クラスも参照してください。
INIFile	このクラスは、初期化ファイルに保存されるデータにアクセスするのに使われます。
Registry	このクラスのメソッドは、Windows レジストリにアクセスするのに使われます。
stdioFile	このクラスは、file クラスから派生し、そのメソッドとプロパティを受け継いでいます。このクラスは、バッファされるストリーム I/O を使って、テキストファイルとバイナリファイルの読み書きの許可を制御するのに使用されます。しかし、このクラスは stdin, stdout, stderr へのストリーム I/O をサポートしません。バイナリファイルへのバッファされない I/O に対する File クラスも参照してください。

4.6 ユーザインターフェースのコントロールクラス

次のクラスは、ユーザインターフェースに関するものです。詳細については、次の Origin ヘルプファイルを参照してください。**Origin C:Origin C Reference:Classes:User Interface Controls** の章を確認してください。

* 印が付いているクラスは、DeveloperKit がインストールされている Origin で利用できます。

クラス	簡単な説明
*BitmapRadioButton	このクラスはビットマップボタンコントロールの機能を提供します。
*Button	このクラスはボタンコントロールの機能を提供します。ボタンコントロールは、クリックでオンオフできる小さな矩形の子ウィンドウです。ボタンをクリックすると、その外観が変わります。標準的なボタンには、チェックボックス、ラジオボタン、プッシュボタンが含まれます。
*CmdTarget	このクラスは、メッセージマップアーキテクチャの基底クラスです。メッセージマップは、コマンドまたはメッセージを記述したメンバー関数に送るのに使われ、メンバー関数はコマンドまたはメッセージを取り扱います。(コマンドはメニュー、コマンドボタン、アクセラレータキーからのメッセージです。) 次の 2 つの主要なフレームワーククラスがこのクラスから派生されます。Window と ObjectCmdTarget。メッセージを取り扱うための新しいクラスを作成するには、これらの 2 つのクラスの一方から新しいクラスを派生するだけです。CmdTarget から直接派生する必要はありません。
*CodeEdit	このクラスは、RichEdit クラスから派生されたものです。これは、コーディングしているテキストでキーワードに対して再定義した色を表示するのに使われます。

*ColorText	このクラスは、DeveloperKit がインストールされている Origin パッケージのみで利用できます。
*ComboBox	このクラスはコンボボックスコントロールを定義するのに使われます。
*Control	このクラスはすべてのコントロールの基本機能を提供します。
*DeviceContext	このクラスは device-context オブジェクトを定義するのに使われます。
*Dialog	このクラスは、画面上でダイアログボックスを表示するための基本クラスです。
*DialogBar	このクラスは、Origin C ベースのダイアログを持つドッキング可能なコントロールを作成するのに使われます。
*DynaControl	このクラスは、編集ボックス、コンボボックス、チェックボックス、ラジオボタンなどのさまざまなタイプのカスタマイズしたインターフェースコントロールを動的に生成します。値は、ツリーノードで保存され、ダイアログ上でツリー構造として表示されます。
*Edit	このクラスは編集コントロールを作成するのに使われます。編集コントロールは矩形の子ウィンドウで、テキストを入力することができます。
*GraphControl	このクラスは、OriginControls, Control および Window クラスから派生し、そのメソッドとプロパティを受け継いでいます。このクラスで定義したメソッドは、ダイアログ内で指定したコントロール内で Origin グラフを表示するのに使うことができます。
GraphObjTool	このクラスは、GraphObjCurveTool の基底クラスです。Origin のグラフウィンドウ上で、関心のある領域でデータを含む矩形を作成し、管理するのに使われます。
GraphObjCurveTool	このクラスは、vectorbase クラスから派生し、そのメソッドとプロパティを受け継いでいます。これらのメソッドとプロパティを使って、Origin のグラフウィンドウ上で、関心のある領域でデータを含む矩形を作成し、管理するのに使われます。このクラスは、コンテキストメニューおよび関連のイベント関数を追加するためのメソッドを提供します。
*ListBox	このクラスはリストボックスを定義するのに使われます。リストボックスは表示と選択に対する文字列項目のリストを表示します。
*Menu	このクラスは、メニューを作成、トラッキング、更新、削除などの取り扱うために使われます。
*OriginControls	このクラスは、ダイアログボックスで Origin ウィンドウを表示するための基底クラスです。
*PictureControl	このクラスは、ダイアログのコントロール内で PictureHolder オブジェクトを塗りつぶすのに使われます。

progressBox	このクラスは、プログレスダイアログボックスを開いて、制御するためのメソッドとプロパティを提供します。プログレスダイアログボックスは、ソフトウェアがデータを処理中であることを示す小さなダイアログボックスです。このダイアログボックスは、処理の進行状況の割合を表示するプログレスバーを含みます。プログレスダイアログは、通常、反復ループに使用します。
*PropertyPage	このクラスは、ウィザードダイアログでプロパティシートの個々のページオブジェクトを作成するのに使われます。
*PropertySheet	このクラスは、ウィザードダイアログでプロパティシートを作成するのに使われます。1 つのプロパティシートオブジェクトには、複数のプロパティページオブジェクトを含むことができます
*RichEdit	このクラスは、テキストフォーマットのためのメソッドを提供します。*rich edit コントロール* はウィンドウで、その中のテキストを記述したり、編集できます。テキストは文字やパラグラフのフォーマットにすることができます。
*Slider	*slider コントロール* は、スライダと任意の刻みを持つウィンドウです。スライダは、マウスまたはキーボードの矢印キーで移動でき、コントロールは通知メッセージを送信して変更を実装します。
*SpinButton	*spin ボタンコントロール* は、コントロールでのスクロール位置や表示している数値などの値を増加または減少させるのに使用する一対の矢印ボタンです。この値は、現在の位置を呼びます。
*TabControl	タブコントロールは、ダイアログの異なるタブの下にある異なる情報を表示するのに使われます。このクラスは、コントロールのグループを表示するのにタブ項目を追加／削除するメソッドを提供します。
waitCursor	ウェイトカーソルは、ソフトウェアがデータを処理中であることを示す視覚的な印です。このクラスは、ウェイトカーソルを開いて、制御するためのメソッドとプロパティを提供します。
*Window	このクラスは、全ての window クラスの基底クラスです。これは MFC の CWnd クラスに似ています。
*WizardControl	このクラスは、ダイアログ内でステップ毎に何かを実装するウィザードのコントロールを作成するのに使われます。このクラスで利用できるメソッドは、ステップを追加/削除することができます。
*WizardSheet	このクラスは、ウィザードダイアログでプロパティシートオブジェクトを作成するのに使われます。プロパティシートには、1 つ以上のページオブジェクトを含めることができます。
*WorksheetControl	このクラスは、OriginControls, Control および Window クラスから派生し、そのメソッドとプロパティを継承しています。このクラスで利用できるメソッドは、ダイアログ内の指定したコントロール内で Origin のワークシートを表示するのに使うことができます。

*WndContainer	このクラスは、派生した制御クラスの基底クラスです。

4.7 Utility クラス

次のクラスについての詳細は、Origin C を参照してください。**Origin C**:**Origin C Reference:Classes:Utility** の章に Origin C のヘルプドキュメントがあります。

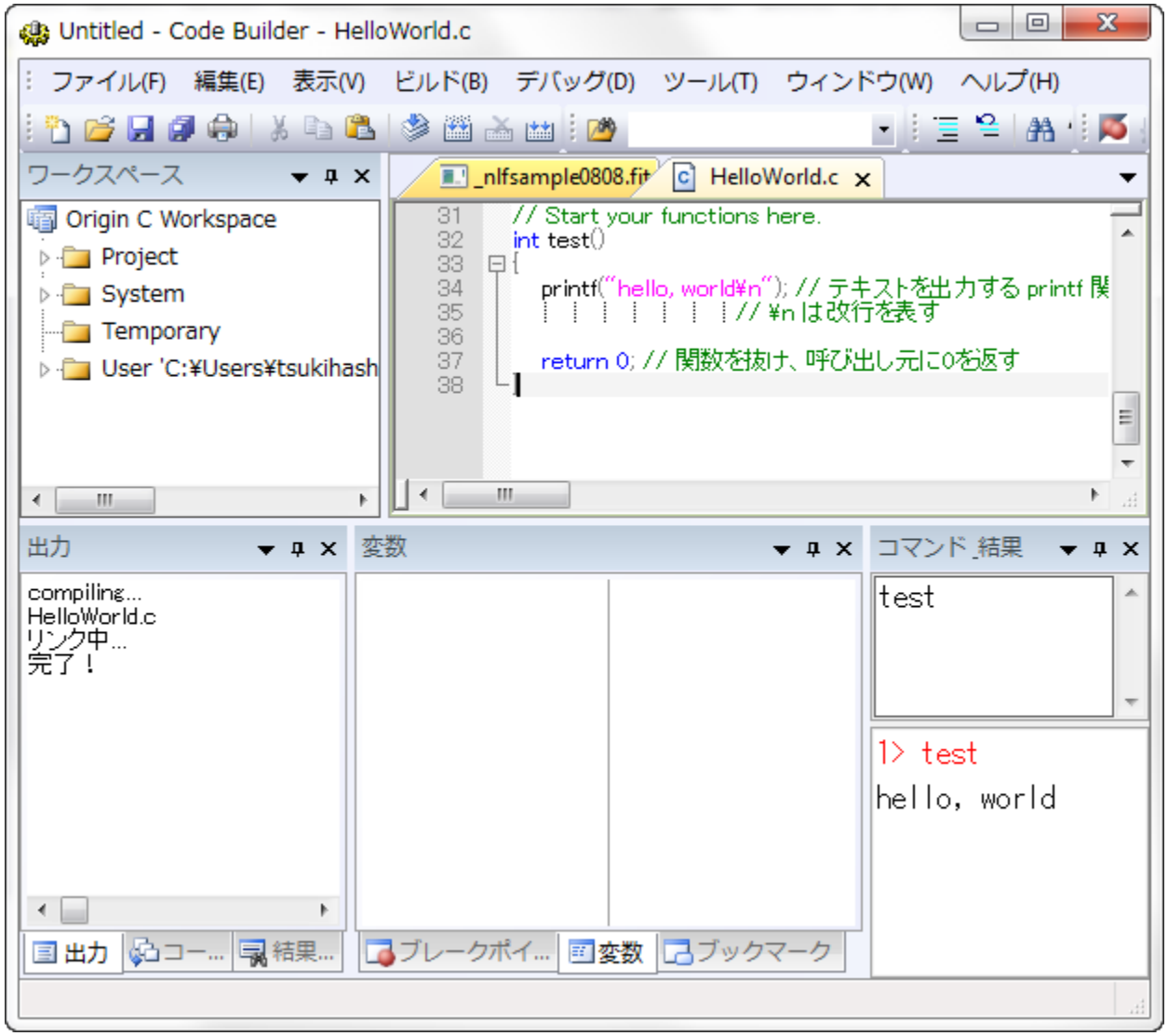
クラス	簡単な説明
Array	このクラスは、ほぼすべてのデータ型とオブジェクトのコレクションです。Array::IsOwner が TRUE のとき、配列はオブジェクトに割り当てられるメモリを所有します。そして配列のサイズが変わったり、破棄されるとオブジェクトは削除されます。
BitsHex	このクラスは、バイトベクター(1 と 0)を 16 進文字列に圧縮したり、16 進文字列をバイトベクターに非圧縮するのに使われます。
Profiler	このクラスは、処理が遅い関数を見つけるため、さまざまな関数の呼び出し時間を測定するのに使われます。

5 Origin C コードの作成と使用

5.1 Origin C ファイルの作成と編集

5.1.1 概要

コードビルダは、Origin C と LabTalk プログラミングの統合開発環境(IDE)です。コードビルダは、Origin C のプログラムを記述／編集したり、コンパイル、リンク、デバッグをしたり、実行するためのツールを提供します。Origin C コードは、テキストエディタで書くことができますが、コードビルダのワークスペースに追加して、コンパイルリンクする必要があります。



コードビルダウィンドウ

5.1.2 ファイルの種類

Origin C は 4 種類のファイル、ソース、オブジェクト、プリプロセス、ワークスペースを利用します。

ソース (*.c, *.cpp, *.h, *.ocz)

ソースファイルは、実際にはテキストファイルで、人間が読むことができる Origin C コードを含みます。コードビルダや他のテキストエディタで作成することができ、どの場所にでも保存できます。コードビルダのテキストエディタは、シンタックスの

色付け、内容に応じたヘルプ、デバッグの機能があります。また、コードビルダでは暗号化 Origin C ファイル(*.ocz)を作成でき、他の人と安全にファイルを共有できます。

ソースファイルはコンパイル、リンク、ロードされるまで、それらを Origin で使用することはできません。

オブジェクト (*.ocb)File, Object File(*.OCB)

ソースファイルがコンパイルされるとき、オブジェクトファイルが生成されます。オブジェクトファイルはソースファイルと同じファイル名を持ちますが、ファイル拡張子は、***.ocb** となります。オブジェクトファイルは、コンピュータが読むことができ、Origin はその関数を呼び出して実行します。Origin は、最初にソースファイルをコンパイルし、そして、ソースファイルが変更されたら再コンパイルします。

Origin 内のオブジェクトファイルは、バージョンに依存し、それらを共有することはできません。関数や Origin C アプリケーションを共有する場合、代わりにプリプロセスファイルを共有します。

プリプロセス (*.op)File, Preprocessed File(*.OP)

デフォルトで、Origin はソースファイルをコンパイルし、オブジェクトファイルを生成します。しかし、下記のシステム変数で、オブジェクトファイルの代わりにプリプロセスファイルを生成するように変更することができます。プリプロセスファイルは、コンパイルが必要ですが、コードの共有に対して次のようなメリットがあります。

- Origin のバージョンに依存しない
- ソースコードを共有せずに関数を共有できる
- ビルド処理がソースファイルよりも高速

オブジェクト(OCB)または、プリプロセス(OP)ファイルのどちらかを生成するシステム変数は、@OCS, @OCSB, @OCSE です。スクリプトウィンドウまたはコードビルダの LabTalk コンソールで、これらの値を変更できます。例えば、スクリプトウィンドウで、次のように入力します。

```
@OCSB=0; // これ以降コンパイルで OP ファイルを作成
```

@OCSCompiler, Generate Pre-processed File

この変数のデフォルト値は 1 で、これは OCB ファイルまたは OP ファイルを作成します。@OCS=0 の場合、コンパイラは OCB ファイルまたは OP ファイルを作成しません。

@OCSBCompiler, Type of Pre-processed File

@OCSB=1 がデフォルト値で、これはコンパイル時にオブジェクトを生成します。OP ファイルを生成するには、OP ファイルがコンパイル時に生成された後、@OCSB=0 とセットします。OP ファイルは、ソースファイルと同じフォルダ内に保存され、同じファイル名を持ちますが、拡張子が OP となります。@OCS=0 の場合、この変数は意味がありません。

@OCSECompiler, Generate Encrypted File

この変数は OroginPro でのみ利用できます。デフォルトの値は 1 です。デフォルトで、コンパイラは実行の詳細を隠した暗号化 OP ファイルを作成します。暗号化 OP ファイルは Origin または OriginPro のバージョンでロードでき、リンクできます。**Note**: この変数は@OCSB=0 のときにのみ意味を持ちます。

Note:

1.生成された OP と OCB は、32bit、64bit のバージョンを持ちます。例えば、32bit バージョンの abc.c から生成された op ファイルは、abc_32.OP という名前になります。

2.Origin 9.0 以降、生成された 32bit あるいは 64bit バージョンのファイルは、対応するバージョン（32bit あるいは 64bit）の Origin でしか動作しません。

ワークスペース (*.ocw)File, Workspace File(*.OCW)

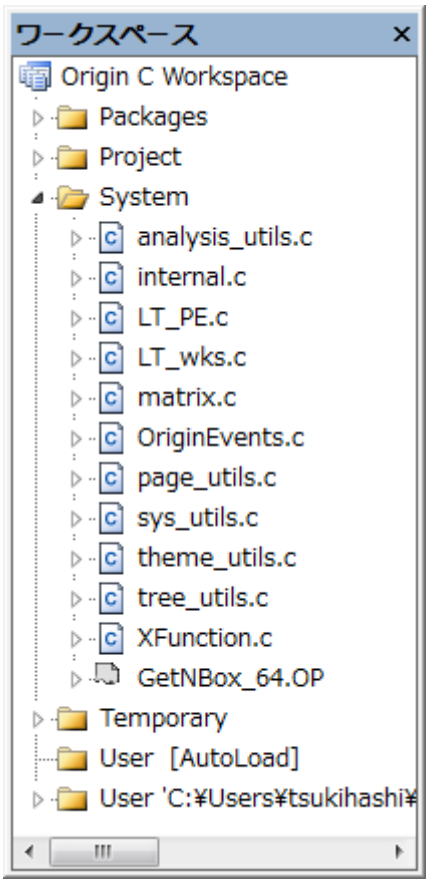
コードビルダでは、数多くの Origin C ソースファイルを含むプロジェクトを作成または使用できます。これらのファイルは、フォルダ内で階層で統合化されていたり、されていない場合があります。このようなファイルを、プロジェクトを切り替える度に、手動でロードするのはとても不便です。

このような理由で、User フォルダに含まれる構造およびファイルはワークスペースファイルに保存することができます。ワークスペースファイルをコードビルダにロードする際に、プロジェクトは直前に保存した状態に戻ります。割り当てられた構造が何であっても、すべてのソースファイルが利用できます。

5.1.3 ワークスペース表示

コードビルダのワークスペース表示は次の、6 つのフォルダがあります。

1. Apps
2. Project
3. System
4. Temporary
5. User [AutoLoad]
6. User



ワークスペース表示

各フォルダ内のファイルは、次のように異なるイベントでコンパイルリンクします。

AppsWorkspace, Packages Folder

このフォルダはパッケージを管理するために使用されます。フォルダのみが含まれ、それぞれユーザファイルフォルダのディスクフォルダを表します。特別なフォルダ Common は全てのパッケージで共有されるファイルを保持するために使用されます。各パッケージフォルダは、"User Files"というフォルダを持ち、これはユーザファイルフォルダ内にあるファイルを持ちます。

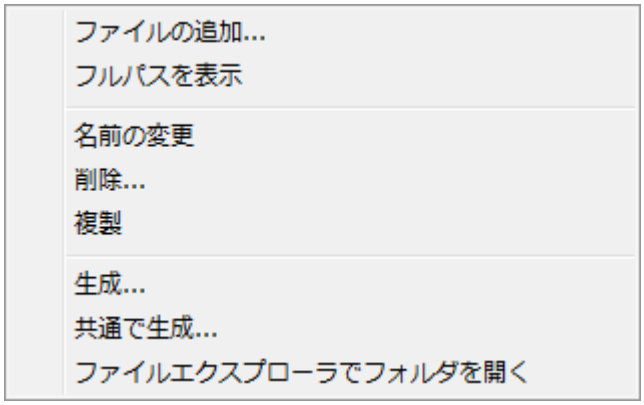
- "Apps" フォルダのコンテキストメニュー

Apps フォルダで右クリックすると、次の 2 つのメニューが表示されます。

1. **既存フォルダの追加...**
2. **新規作成**

最初のメニューは、ユーザファイルフォルダの既存フォルダを選択するのに使用し、2 つ目のメニューは、**Untitled** という名前の新しいフォルダを作成して、ユーザファイルフォルダにも **Untitled** という新しいディスクフォルダを作成します。**新規作成**メニューを繰り返し選択すると **Untitled** フォルダが複数作成されます。

- **Common** 以外の各パッケージフォルダのコンテキストメニュー



1. ファイルの追加

フォルダにファイルを追加するのに使用します。各パッケージフォルダは、ユーザファイルフォルダのディスクフォルダを表します。追加されたファイルがユーザファイルフォルダからのものである場合、それがインストールされていることを示すために、ファイルは、"User Files" サブフォルダに置かれます。選択されたファイルがまだユーザファイルフォルダ、あるいは、ユーザファイルフォルダ\packageFolder\にない場合、ユーザファイルフォルダ\packageFolder\ フォルダにコピーされます。

2. フルパスを表示

ファイルのフルパスを表示/非表示にします。

3. 名前の変更

パッケージフォルダの名前を変更します。

4. 削除

パッケージフォルダを削除します。ディスクフォルダがある場合、ディスクフォルダとファイルをともに削除するか聞かれます。

5. 複製

パッケージフォルダとそのファイルを複製します。

6. 生成

パッケージマネージャを起動し、パッケージフォルダからファイルを追加します。同じ名前の OPX ファイルがユーザファイルフォルダユーザファイルフォルダにある場合、パッケージフォルダから全ファイルが追加する前に、OPJ がロードされ、すべてのファイルが削除されます。

7. Common で生成

生成と同じですが、Common フォルダ内のファイルも追加されます。

Note:Common はパッケージではないので、コンテキストメニューは最初の 2 つの項目だけです。各パッケージ内にある **User Files** フォルダも同様です。

ProjectWorkspace, Project Folder

Project フォルダ内のファイルは、現在の Origin プロジェクトファイル(*.OPJ)に保存されます。それらを含む Origin プロジェクトファイルを開く際に、それらはコードビルダワークスペースのプロジェクトフォルダに追加されます。それらは自動的にコンパイルリンクされ、プロジェクトファイルを開きます。

SystemWorkspace, System Folder

System フォルダ内のファイルは、Windows フォルダ内に外部的に保存されます(通常 Origin C フォルダまたはそのサブフォルダの 1 つ)。それらは、自動的にコードビルダワークスペースの System フォルダに追加され、Origin が起動したときにコンパイル、リンクされます。

TemporaryWorkspace, Temporary Folder

Project, System, User フォルダ内にリストされていないすべてのファイルは、Origin を使用しているときにロードされ、コンパイルされ、Temporary フォルダに現れます。例えば、グラフをエクスポートする場合、グラフエクスポートを取り扱うために使われるすべてのファイルは、Temporary フォルダに現れます。

User [AutoLoad]Workspace, User Folder

このフォルダは、次で説明する **User Folder** に似ています。相違点としては、手動でのコンパイルリンクが必要ないことです。このフォルダ内のファイルは、Origin 起動時に自動的にコンパイルリンクされ、このフォルダ下のファイルで定義された関数が利用可能になります。

UserWorkspace, User Folder

User フォルダ内のファイルは Windows フォルダ内に外部保存され、コードビルダワークスペースの User フォルダに手動で追加されます。コンパイルとリンクは、ユーザがコードビルダで行います。

Note:

Apps と **User [AutoLoad]** フォルダの内容は、**Project** フォルダの内容が各プロジェクトファイル（OPJ）で固有な間は、Origin のセッションをまたいで保持されます。

5.1.4 コードビルダのクイックスタート Code Builder

コードビルダの操作について簡単に説明します。

1. キーボードで Alt+4 を押すか、コードビルダボタン をクリックして、コードビルダを開きます。
2. Ctrl-N、または新規ツールバーボタンをクリックして、新しいソースコードファイルを作成します。新規ファイルダイアログが開いたら、ソースコードファイルの名前を入力し、Enter キーを押し、OK ボタンをクリックします。
3. エディタウィンドウが開きます。エディタウィンドウの最後の行に行き、Enter キーを押して、新しい空の行を開始します。そして、次の関数を入力します。

```
void HelloWorld()
{
        printf("Hello World, from Origin C¥n");
}
```

4. この関数を呼び出す前に、コードをコンパイルリンクする必要があります。Shift+F8 キーを押すか、ツールバーのビルドボタン をクリックします。
5. 出力ウィンドウにコンパイルリンクの処理が表示されます。エラーが表示されたら、関数をチェックし、エラーを訂正します。エラーがなければ、関数を呼び出す準備ができています。
6. コマンド-結果ウィンドウの上部をクリックします。関数名を入力し、Enter キーを押します。コマンド-結果ウィンドウの下部で、関数名、入力した行が表示され、それに続いて関数の出力行が表示されます。

これらのステップは、コードビルダで行うことができますが、Origin C ファイルを効率よく記述したり、デバッグや実行の助けとなる詳細設定があります。これらについては、後に続くセクションで説明しています。

5.2 コンパイル、リンク、ロード

Origin C 関数にアクセスする前に、コードビルダでコンパイルしてリンク(ビルド)する必要があります。

エラー無く関数をコンパイルリンクしたら、自動的にロードされ、現在の Origin セッションでアクセスできます。今後の Origin セッションで関数にアクセスするには、再ロードし、リンクする必要があります。この処理は素早く、自動化することができます。

この章では、Origin C のソースファイルとプリプロセスファイルを手動または自動でビルドする方法について説明します。

5.2.1 コンパイルとリンク Compile and Link Source Files

Origin C のソースファイルまたはプリプロセスファイルで定義した関数を作成するため、以下のステップを行う必要があります。

- コードビルダワークスペースにファイルを追加
- ファイルをコンパイル

- すべての関連ファイルをリンクし、必要に応じて関連ファイルもコンパイルして、作成したオブジェクトファイルをロード

すべてのファイルをコンパイルし、リンクする処理はビルドとして扱われます。

ワークスペースにファイルを追加 Workspace, Load Files

ソースファイルまたはプリプロセスファイルをコンパイルリンクする前に、ファイルは、**コードビルダ**ワークスペースフォルダである Project, User, System, Temporary の内の 1 つに追加する必要があります。すべてのソースファイルは、最初に作成されるか、User フォルダにロードされます。

ファイルをコンパイルする

ファイルをワークスペースに追加した後、(**コンパイルボタン** をクリックして)コンパイルし、オブジェクトファイルを生成する必要があります。オブジェクトファイルは、ソースファイルまたはプリプロセスファイルと同じ名前で、**OCB** という拡張子を持ちます。Origin8.1 以降、オブジェクトファイルは、**アプリケーションデータ**フォルダに保存されます。古いバージョンでは、ファイルは **User Files\OCTemp** フォルダに保存されます。

ワークスペースをビルドする

アクティブなファイルとすべての関連ファイルをビルドするには、**ビルド** ボタンを選択するか、**すべてをリビルド** ボタンを選択し、ワークスペース内のすべてのファイルをビルドします。作成したオブジェクトファイルは、自動的にメモリにロードされ、リンクされ、ファイルに定義した関数は Origin の中で実行可能になります。

オブジェクトファイルが生成されると、その後のビルド処理がより高速になります。ソースファイルとプリプロセスファイルに変更が無ければ、**コードビルダ**はオブジェクトファイルを直接ロードし、リンクしますが、ファイルをリビルドしません。

ビルドとすべてをビルドの違い

ビルド:与えられたフォルダがコードビルダでアクティブフォルダで、ビルドツールバーボタン がクリックされると、そのフォルダ内のすべてのファイルがコンパイルリンクされます。

すべてをビルド:コードビルダのすべてをリビルドボタン がクリックされると、コードビルダの ***すべて*** のフォルダ内のファイルがコンパイルリンクされます。

5.2.2 ビルドの自動化

最初に、すべての Origin C ソースファイルまたはプリプロセスファイルが User フォルダ内で作成され、開かれます。そして、上記で述べたように Origin C ソースファイルを手動でビルドします。しかし、ビルド処理を自動化するとメリットがあります。これは、コードビルダのフォルダ構造を使用することで行なったり、機能的に少しの違いがあったり、**起動時のビルド**オプションを利用して、行うことができます。

Project フォルダにファイルを追加する Building Origin C File with Project

コードビルダの **Project フォルダ**にファイルを追加すると、関連の Origin プロジェクトが開いたときに、自動的にビルドされます。

次の方法で、**Project** フォルダにファイルを追加できます。

- Project フォルダを右クックして、**ファイルの追加**を選びます。
- 別のワークスペースフォルダからファイルをドラッグし、Project フォルダにドロップします。

[AutoLoad]フォルダにファイルを追加する Building Origin C Files under Workspace System Folder

以前は、コードビルダの **System** フォルダを右クリックして**ファイルを追加**できました。Origin 2015 以降、このフォルダにユーザーファイルを追加することはできません。Origin の起動時に自動的にファイルをコンパイルリンクさせるには、これらのファイルを **[Autoload]**フォルダに追加する必要があります。

[AutoLoad]フォルダにファイルをついかするには次のようにします。

- [AutoLoad] フォルダを右クリックして、**ファイルの追加**を選択します。
- 他のワークスペースフォルダからファイルをドラッグして、[AutoLoad]フォルダにドロップします。

Origin 起動時にワークスペースをビルド Building on Startup

起動時にビルドオプションは、Origin 起動時に直前に開いたコードビルダワークスペースをビルドします。

Origin が起動したときに、Origin C ワークスペースの **System** フォルダの内容を確認し、変更が見つかったら、コンパイルリンクしようとします。また、**起動にビルド**オプションを有効にして、**ユーザ**フォルダ内のファイルにこの手順を行うことができます。

1. コードビルダを実行します。
2. ワークスペースが見えていない場合、**表示**メニューの**ワークスペース**を選びます。
3. **Origin C ワークスペース**を右クリックします。
4. **起動にビルド**の項目がチェックされていない場合、クリックします。

次回 Origin を起動すると、User フォルダ内のファイルがチェックされ、変更があったファイルをコンパイルリンクします。

Origin 起動時にソースファイルをビルド

次の操作方法で、起動時に Origin C ソースファールをロードし、コンパイルするために Origin.ini ファイルを編集します。

1. Origin が開いていたら閉じます。Origin.ini ファイルを開きます。このファイルはユーザファイルフォルダにあります。コマンドウィンドウで %Y= を実行すると、ユーザファイルフォルダの場所が確認できます。ファイルが見つからない場合は、Origin インストールフォルダに移動します。
2. **[Config]** セクションで、OgsN = OEvents の行のセミコロン(;)をとり、コメントとして扱うのをやめます。N は、使用されている数字でなければ、何でも構いません。変更後、保存してファイルを閉じます。
3. Origin のインストールフォルダにある、OEvents.ogs を開きます。**[AfterCompileSystem]** セクションに以下の行を追加します。

```
run.LoadOC(Originlab¥AscImpOptions, 16);
```

4. 変更後、保存してファイルを閉じます。

5. Origin を再度起動し、コードビルダを開きます。Temporary フォルダに 3 つのファイルがあります。AscImpOptions は、fu_utils.c と Import_utils.c に依存するので、コンパイラはこれら 2 つのファイルを一緒にコンパイルします。詳細は LabTalk ヘルプで、**run.LoadOC** を検索して確認してください。

または、ワークスペースの ***User [AutoLoad]***フォルダを使用してください。フォルダに追加されたファイルは、Origin の起動時に自動的にロードされます（上記参照）。

5.2.3 スクリプトでビルドする Building with Dependent Files

LabTalk スクリプトで Origin C を呼び出したい場合、ソースファイルのコンパイルとリンクが完了していることを確認する必要があります。その後、LabTalk コマンド *Run.LoadOC* を使用して、特定のソースファイルをコンパイルし、リンクします。例えば、

1. コードビルダで、ファイル: 新規ワークスペースを選択して新しいワークスペースを作成します。ここでは、Temporary フォルダは空です。
2. コマンドウィンドウで、次のスクリプトを実行します。すると *dragNdrop.c* ファイルとその関連ファイルすべてが Temporary フォルダにロードされ、コンパイルされます。

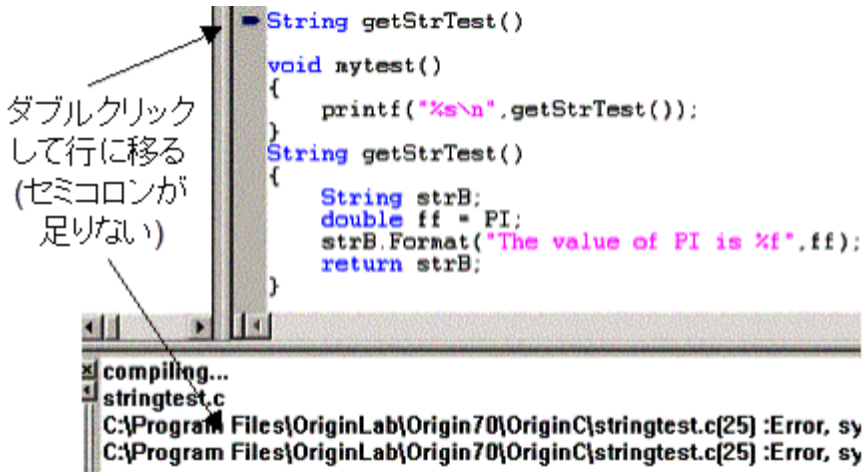
```
if(run.LoadOC(OriginLab¥dragNdrop.c, 16) != 0) { type "Failed to load dragNdrop.c!"; return 0; }
```

5.2.4 エラーを識別する Building, Handle Compiling and Linking Error

コードビルダでソースファイルをコンパイルリンクするとき、コンパイルおよびリンクの結果はコードビルダの**出力**ウィンドウに表示されます。

コンパイルリンクが問題なく行われた場合、**出力**ウィンドウにはソースファイルがコンパイルされたことが表示されます。**完了!** と表示され、問題がなかったことが分かります。

コンパイルリンク中にエラーが発生したら、**出力**ウィンドウには、ファイル名、行番号、エラーが発生したことが表示されます。**出力**ウィンドウ内のエラー行をダブルクリックすると、ソースファイルをアクティブにし、エラーを含むコード行にカーソルを移動します。



5.3 デバッグする

5.3.1 コードビルダでデバッグする

コードビルダ には、Origin C や LabTalk のコードをデバッグする機能があります。ブレークポイントをセットしたり、解除でき、関数にステップイン、ステップアウトしてコードを 1 度に 1 行ずつ実行したり、変数内の値を監視することができます。デバッグはデフォルトで有効になっています。デバッグメニューのブレークポイントを有効にするを使ってデバッグをオンまたはオフにすることができます。メニュー項目の隣にチェックボックスがある場合、デバッグ機能はオンです。

5.3.2 デバッグのマクロ

Origin C は、多くの用途を持つ複数パラメータを定義することができます。多くのプログラマは、プログラムの流れを指示するコードを開発し、変数の値を表示するコードを開発する際に、出力ステートメントを使います。

出力マクロを作成する

便利なデバッグのテクニックは、出力マクロを定義し、そのマクロを下図のように配置することです。

```
#define    DBG_OUT(_text, _value)     out_int(_text, _value);



void DebugStatements()

{

     int ii;

     DBG_OUT("ii at t0 = ", ii)

     ii++;

     DBG_OUT("ii at t1 = ", ii)

     ii++;

     DBG_OUT("ii at t2 = ", ii)

     ii++;

     DBG_OUT("ii at t3 = ", ii)

     printf("Finished running DebugMacros.");

}
```

デバッグマクロの本体をコメントアウトする

開発期間中に、マクロの本体を上記で定義したようにしたままにすると、デバッグメッセージがメッセージボックスに表示されます。しかし、開発が完了したら(または安定的な動作になったら)、マクロを下記のように再定義し、デバッグステートメントを非表示にすることができます。

```
#define    DBG_OUT(_text, _value)    //  out_int(_text, _value);
```

DBG_OUT マクロの本体をコメントアウトし、リビルドすると、その使用の可能なインスタンスを除去せずに、将来再利用するためにそれらを保存して、デバッグステートメントを非表示にすることができます。コードは再び修正したり、デバッグする必要はなく、マクロの本体を単に非コメントにするだけです。

5.4 コンパイルした関数を使用する

Origin C 関数をコンパイル、リンク、ロードしたら、Origin で使用する準備ができています。これは名前で関数を呼び出し、LabTalk スクリプトコマンドを受け付ける Origin のどの場所からでも必要な引数を提供できるということです。スクリプトウィンドウ、コマンドウィンドウ、Origin GUI 内のカスタムボタンなどから関数を呼び出せます。LabTalk スクリプトガイドのスクリプトの実行の章 では、Origin C 関数を使用することができる Origin 内のすべての場所についての詳細があります。

5.4.1 LabTalk スクリプトから Origin C 関数にアクセスする

Origin C 関数は、他の Origin C や LabTalk スクリプトから呼び出すことができます。このセクションでは、Origin C 関数を LabTalk からアクセスする方法について説明しています。

Origin C コードから LabTalk にアクセスすることについての情報は、LabTalk にアクセスする の章を参照してください。

LabTalk から Origin C 関数にアクセスする

関数定義の前に Origin C コードに pragma ステートメントを配置して、Origin C コードにアクセスする LabTalk を制御できます。

```
#pragma labtalk(0) // LabTalk で OC 関数を無効に

void foo0()

{

}



#pragma labtalk(1) // LabTalk で OC 関数を有効に（デフォルト）

void foo1()

{

}



#pragma labtalk(2) // LabTalk コマンド '''run -oc''' が必要

void foo2()

{

}
```

上記のコードは、foo0 を LabTalk からの呼び出しを禁止し、foo1 は LabTalk から呼び出すことができ、foo2 は、**run -oc** コマンドを使って、LabTalk から呼び出すことができます。2 番目の pragma をコメントアウトしていると、foo0 と foo1 の両方が LabTalk から呼び出すことができません。これは、1 つの pragma ステートメントが pragma の後のすべての関数に適用され、次の pragma またはファイルの最後まで続きます。

Origin C 関数にアクセスする LabTalk を制御する LabTalk のシステム変数もあります。変数は **@OC** で、そのデフォルトは 1 であり、アクセス可能ということです。変数を 0 にセットするとアクセス不可となります。

LabTalk ぁら呼び出せる関数を一覧表示する

LabTalk の list コマンドは、LabTalk から呼び出すことができる Origin C 関数のすべての名前を出力するのに使用できます。オプションを使って、関数を一覧表示する形式を変更することができます。

```
list f;  // LabTalk から呼び出し可能な関数一覧

list fs; // string を返す関数の一覧

list fv; // vector を返す関数の一覧

list fn; // numeric を返す関数の一覧

list fo; // void を返す関数の一覧
```

@OC=0 という設定は、Origin C 関数を LabTalk で非表示にし、そのため **list f** コマンドは結果を表示しません。

関数に引数を渡す

LabTalk スクリプトは Origin C の内部で使用しているすべてのデータ型をサポートしているわけではありません。次の表は、与えられた引数(戻り)の型で Origin C 関数を呼び出すときに、渡される(返される)LabTalk 変数の型 です。最後の列は、引数の型が参照によって渡すことができるかどうかを示しています。

Origin C	LabTalk	参照渡し
int	int	可
double	double	可
string	string	可
bool	int	不可
matrix	matrix range	可
vector<int>	dataset	可
vector<double>	dataset	可
vector<complex>	dataset	不可
vector<string>	dataset, string array*	不可

* 文字列配列は参照渡しができません。

この表が示すように、**string**, **int**, **double** 型の Origin C 関数の引数は LabTalk から値または参照で渡すことができます。しかし、Origin C 関数は実行時に渡される型で記述しなければなりません。

値で渡す

以下は、LabTalk から Origin C に値で引数を渡すサンプルです。各サンプルに対するフォーマットは、Origin C 関数の宣言の行を与え、LabTalk のコードを使って呼び出されます。Origin C 関数の本体は、変数を渡すというサンプルに対しては重要ではないので、除外しています。

関数の単純なケースは、引数を受け付け、double 型の引数を受付け、double 型を返します。

```
double square(double a)  // Origin C 関数の宣言

double dd = 3.2;         // LabTalk 関数の呼び出し

double ss = square(dd);

ss =;                    // ss = 10.24
```

ここで、データタイプに割り当てるデータセット変数または範囲を使って、vector 引数を取り、vector を返す Origin C 関数が LabTalk から呼ばれます。

```
vector<string> PassStrArray(vector<string> strvec)
```

LabTalk から 3 つの方法で呼び出すことができます。

```
dataset dA, dB;

dB = Col(B);

dA=PassStrArray(dB);

Col(A)=PassStrArray(Col(B));  // または Col を直接使い、Col = dataset

// または、LabTalk 範囲が使われる

range ra = [Book1]1!1, rb = [Book1]1!2;

ra = PassStrArray(rb);
```

参照で引数を渡す

以下の Origin C 関数に対して、引数宣言内のアンパサンド **&** の文字は、引数が参照で渡されていることを示しています。

```
double increment(double& a, double dStep)

double d = 4;

increment(d, 6);

type -a "d = $(d)";  // d = 10
```

次は、参照で渡すいくつかの引数とそれ以外を値で渡すサンプルです。

```
int get_min_max_double_arr(vector<double> vd, double& min, double& max)
```

```
dataset ds = data(2, 30, 2);
double dMin, dMax;
get_min_max_double_arr(ds, dMin, dMax);

//または列からデータセットを使う、Col(A)にデータがあること
get_min_max_double_arr(Col(A), dMin,  dMax);
```

次のサンプルは、LabTalk の matrix range 変数を Origin C 関数に参照で渡すことを示しています。

```
// データを vector から matrix にセット
void set_mat_data(const vector<double>& vd, matrix& mat)
{
        mat.SetSize(4,4);
        mat.SetByVector(vd);
}

range mm = [MBook1]1!1;
dataset ds = data(0, 30, 2);
set_mat_data(ds, mm);
```

同じ名前を持つ関数の優先規則

ユーザ定義またはグローバルな Origin C 関数が組み込みの LabTalk 関数と同じ名前を持つとき、LabTalk の vecotr 表記を使う時を除いて、Origin C 関数が優先順位が高くなります。

優先順位:

1. LabTalk 関数 (vector)
2. Origin C 関数
3. LabTalk 関数 (scalar)

通常の LabTalk 関数(値の範囲を戻し、vector 表記で使われる)は、同じ名前の Origin C 関数よりも優先順位が高くなります。それ以外の場合には、Origin C 関数が呼ばれます。

5.4.2 値の設定ダイアログの関数を定義する

列または行列のどちらかの**値の設定**メニューで現れるダイアログボックスで、Origin C を使って関数を定義することができます。

Origin C 関数が Origin プロジェクトの一部としてビルドされると、---コードビルダの Project または System に自動的に配置されるか、user フォルダ内の関数を手動でビルドするかのいずれか---(列と行列の両方)の**値の設定**ダイアログの **F(x)** の **ユーザ定義** セクションで利用できます。**F(x)** メニューの異なるセクションに関数を割り当てるには、関数ヘッダの一部として、新しいセクション名を含む pragma を発行します。例えば、次のコードは、Math セクションに関数 **add2num** を追加し、統計セクションに関数 **mean2num** を追加します。

```
#pragma labtalk(1,Math)

double add2num(double a, double b)

{

          return a + b;

}



#pragma labtalk(1,Statistics)

double mean2num(double a, double b)

{

          return (a + b)/2;

}
```

この方法で、多くの関数を 1 つのソースファイル内で定義し、ビルドし、**F(x)** メニューの目的の場所で直ちに利用できます。

F(x) メニューに追加される関数は、次の制約に従います。

- 関数の戻り型は、void 型にすることはできません。
- 関数は引数の型に対して、参照またはポインタを持つことはできません。

5.5 Origin C コードを配布する

5.5.1 ソースコードを配布する Distributing CodeShare Code

Origin ユーザは、ソースファイル(.C、.CPP、.OCZ)またはプリプロセスファイル(.OP)のいずれかを配布することで、他の人と Origin C ソースコードを共有することができます。

他の人がアプリケーションのソースコードを見る必要がなければ、ソースファイル(.C または.CPP)ではなく、暗号化 Origin C ファイル(.OCZ)かプリプロセスファイル(.OP)を配布することをお勧めします。

暗号化 OCZ ファイルは、コードビルダにドラッグアンドドロップするだけで内容の表示や編集が可能です。最初ファイルを開こうとしたときにパスワードを聞かれますが、同じ Origin セッション内では一度しか聞かれません。

詳細は、**Origin C ファイルの作成と編集**セクションのファイルの種類を確認してください。

暗号化された OCZ ファイルを Origin 内で開くときには、Origin 2016 SR0 移行のユーザは暗号化されていない*.c ファイルもしくは*.cpp ファイルとして*.ocz ファイルを再保存します。その際は、**ファイル**:名前を付けて保存を選択し、ファイルの種類ドロップダウンリストからファイルの種類を選びます。

5.5.2 アプリケーションを配布する

アプリケーションを作成したら、1 つのパッケージファイルとして、他のユーザにアプリケーションを配布することができます。

パッケージ・マネージャ を使って、すべてのアプリケーションファイルを 1 つのパッケジファイル(.OPX)にまとめることができます。Package Files アプリケーションファイルをパッケージに追加するときには、プリプロセスファイル(.OP)またはソースファイル(.C または.CPP)を追加することに注意してください。両方を追加する必要はありません。

ユーザは、パッケージファイルを Origin に直接ドロップしてインストールすることができます。

以下は、すべてのアプリケーションファイルを 1 つの OPX にパッケージする方法を示すサンプルです。ユーザは、パッケージファイルを Origin にドロップし、ボタンをクリックしてソースファイルを実行することができます。

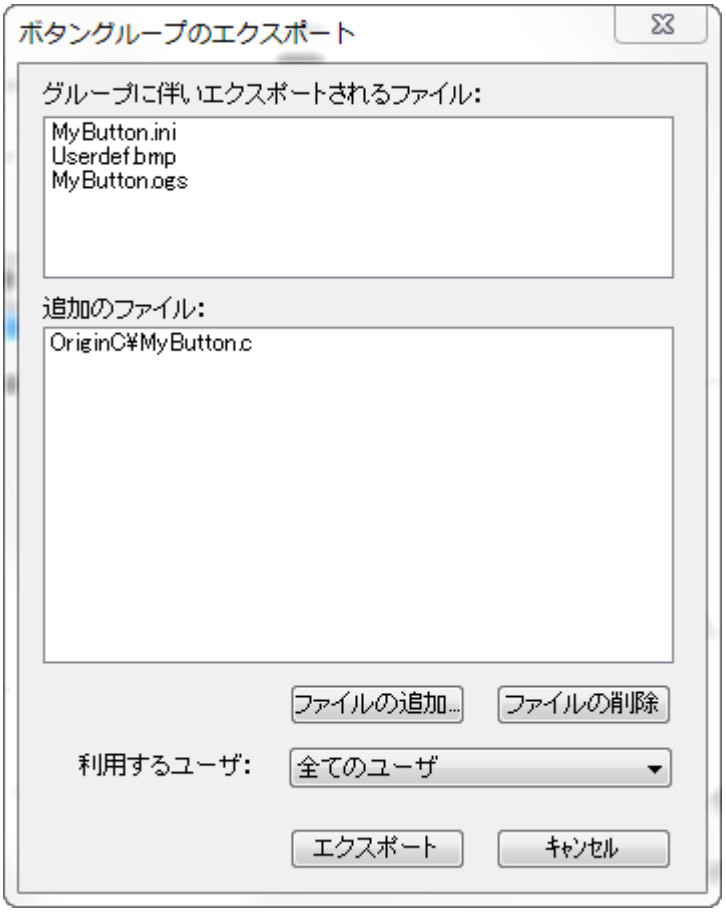
1. Origin C ファイルを準備します。**コードビルダ**で、**ファイル**:新規作成メニューから *MyButton.c* という新しい c ファイルを作成し、以下のコードをそこにコピーして、*ユーザファイルフォルダの OriginC* サブフォルダに保存します。

```
void OnButtonClick() { Worksheet wks = Project.ActiveLayer();   DataRange dr; dr.Add(wks, 0, "X");
dr.Add(wks, 1, "Y");   GraphPage gp; gp.Create(); GraphLayer gl = gp.Layers(0);   int nn =
gl.AddPlot(dr); gl.Rescale(); }
```

2. *MyButton.ogs* という OGS ファイルを作成し、Origin C ソースコードをロードし、関数を呼びます。以下のコードをコピーし、ユーザファイルフォルダに保存します。

```
[Main] if(0 == Run.LoadOC(%Y¥OriginC¥MyButton.c)) { OnButtonClick; }
```

3. Origin メニューから**表示:ツールバー**を選択します。**カスタマイズ**ダイアログで、**ボタングループ**タブを選び、**新規作成**をクリックして、**ボタングループの作成**ダイアログを開きます。**グループ名**として *MyButton* をセットし、**ボタンの数**は 1 のままにし、**ビットマップ**にはユーザファイルフォルダから *Userdef.bmp* ファイルを選び、OK ボタンをクリックします。**名前を付けて保存**ダイアログで、保存ボタンをクリックして、*MyButton.ini* ファイルをデフォルトパスに保存します。
4. **カスタマイズ**ダイアログで、**グループ**リストから *MyButton* を選んで、**ボタン**パネルから ボタンをクリックして選び、設定をクリックして**ボタン設定**ダイアログを開きます。**ファイル名**に *MyButton.ogs* を選び、**セクション名**に Main と入力し、次のチェックが外れていることを確認してください:**行列**、**Excel**、**グラフ**、**レイアウト**。OK をクリックしてダイアログを閉じます。
5. **エクスポート**をクリックし、**ボタングループのエクスポート**ダイアログで、**ファイルの追加**をクリックして *MyButton.c* ファイルを選択します。



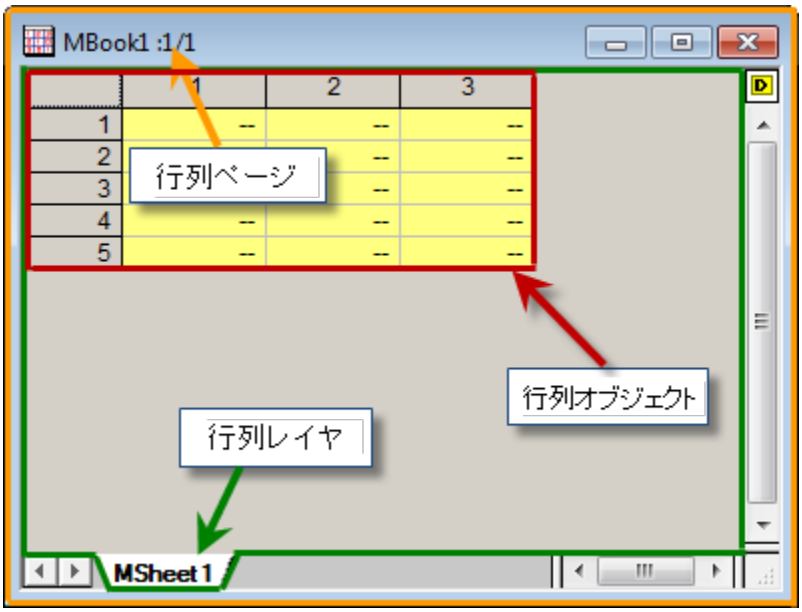
6. **エクスポート**をクリックし、名前を付けて保存ダイアログで、保存をクリックして指定したフォルダに *MyButton.OPX* ファイルを保存します。
7. メニューから**ツール**:パッケージ・マネージャーを選び、開いたダイアログボックスで、**ファイル**:開くを選択して *MyButton.OPX* を開きます。以下のスクリプト

```
Run.LoadOC(%Y¥OriginC¥MyButton.c);
```

8. を、**LabTalk スクリプト**:**インストール後**に入力して OriginC のソースファイルをロードします。Origin に OPX ファイルをドロップしてこのアプリケーションをインストールするときに、このスクリプトが実行します。

6 行列ブック、行列シートと行列オブジェクト

MatrixPage クラスは、Origin 行列ブックを操作するものです。各行列ブックは、MatrixLayers のコレクションを含み、各行列レイヤは、MatrixObjects のコレクションを含みます。



6.1 行列ブックの基本操作

Origin C の MatrixPage クラスは、Origin の行列ブックに共通するメソッドとプロパティを提供します。このクラスは、Page クラス由来のもので、そのメソッドとプロパティを継承します。そして、行列ブックは Origin の WorksheetPage と同じデータ構造レベルに属し、どちらもウィンドウです。そのため、これらの操作は多くの面で似通っています。

6.1.1 ワークブックのような操作

行列ブックとワークブックの操作は似通っていて、多くの部分で共通しているので、ワークブックの基本操作の章を利用できます。

1. 新しい行列ブックを作成

   同メソッド Create を使用します。

```
MatrixPage matPg;

matPg.Create("Origin"); // Origin テンプレートを使用して行列ブックを作成
```

2. 行列ブックを開く

   Open メソッドにより行列ブックを開く場合の違いは、行列ブックの拡張子が *ogm* であることです。

3. 行列ブックにアクセスする

既存の行列ブックにアクセスする方法は複数あり、メソッドはワークブックと同様です。Project クラスは、プロジェクト内の全行列ブックのコレクションを含みます。次のサンプルでは、それらをどのようにループするかを示します。

```
foreach(MatrixPage matPg in Project.MatrixPages)

    out_str(matPg.GetName()); // 行列ブック名を出力
```

また、Collection クラスの Item メソッドに、行列ブックのインデックスを渡すことによって行列ブックにアクセスできます。

```
MatrixPage matPg;

matPg = Project.MatrixPages.Item(2);

if(matPg) // 3 番目の行列ブックがあるとして

    out_str(matPg.GetName()); // 行列ブック名を出力
```

行列ブック名がわかっている場合、クラスコンストラクタにその名前を渡すことで、行列ブックにアクセスできます。

```
MatrixPage matPg("MBook1");

if(matPg) // "MBook1"という行列ブックがある場合

    matPg.SetName("MyBook1"); // 行列ブックの名前を変更
```

4. 行列ブックを保存

メソッド SaveToFile は、行列ブックを *.ogm ファイルとして保存するのに使用されます。

```
MatrixPage matPg("MBook1");

// 行列ブックを OGM ファイルとして保存

bool bRet1 = matPg.SaveToFile("D:¥¥"+ matPg.GetName() + ".ogm");
```

5. 行列ブックの表示/非表示

これは、OriginObject クラス由来の Show プロパティによるワークブックの表示/非表示と同じです。

6. 行列ブックをアクティブにする

行列ブックをアクティブにするには、ワークブックと同じように PAGE_ACTIVATE 値のパラメータを沙汰することにより SetShow メソッドを使用できます。

```
MatrixPage matPg("MBook1");

matPg.SetShow(PAGE_ACTIVATE);  // 行列ブックをアクティベート
```

7. 行列ブックを削除

   行列ブックの削除にも、Destroy メソッドが使用できます。

```
MatrixPage matPg;

matPg = Project.MatrixPages.Item(0); // プロジェクト内の最初 j の行列を取得

if( matPg ) // 行列ブックがある場合

    matPg.Destroy(); // 行列ブックを削除
```

8. 行列ブックを複製

   Clone メソッドを使用して行列ブックを複製します。

```
// データとスタイル保持して "MBook1" ウィンドウを複製

// 呼び出す前にこれらのウィンドウが存在することを確認

MatrixPage matPage("MBook1");

MatrixPage matPage1 = matPage.Clone();
```

9. 行列ブックの名前とラベル

   行列ブックのショートネーム、ロングネーム、コメントを扱うために、Origin C は、ワークブックの扱いと同様に、受け継がれたメソッドを含む SetName, SetLongName, SetComments, Label プロパティを提供します。

6.1.2 イメージサムネールを表示する

イメージサムネールを表示または非表示にするには、MatrixPage::ShowImageThumbnails メソッドを使用出来ます。

```
MatrixPage mp("tangent");

mp.ShowImageThumbnails(true);  // サムネールを有効にするために true をパス
```

6.2 行列シート

MatrixLayer クラスは、行列シートを操作するために使用されます。

6.2.1 行列シートの基本操作

行列シートとワークシートは、Origin オブジェクト構造でのレベルが同じため、このセクションのサンプルは、ワークシートの基本操作の章で紹介しているものと似通っています。OCGuide:Worksheet Basic Operation

新しい行列シートを追加する

AddLayer メソッドを使用して、行列ブック内の行列シートを追加します。

```
// "MBook1"という名前の行列ブックにアクセス

MatrixPage mp("MBook1");



// 行列ブックに新しいシートを追加

int index = mp.AddLayer("New Matrix Sheet");



// 新しい行列シートにアクセス

MatrixLayer mlayerNew = mp.Layers(index);
```

行列シートをアクティブにする

行列ブック内のシートをアクティブにするには、関数 set_active_layer を使用できます。

```
// フルネームで行列シートにアクセス

MatrixLayer mLayer("[MBook1]MSheet1");



// この行列シートをアクティブに設定

set_active_layer(mLayer);
```

行列シートを削除する

Destroy メソッドを使用して行列シートを削除します。

```
MatrixLayer ly = Project.ActiveLayer();

if( ly ) // アクティブレイヤが行列シートの場合

    ly.Destroy(); // 行列シートを削除
```

行列ブック内の行列シートにアクセスする

ワークブック内のワークシートにアクセスするのと同様、行列ブック内の行列シートへは以下の方法によりアクセスできます。

1. 全レイヤ名による

```
// 行列シートのフルネーム

string strFullName = "[MBook1]MSheet1!";



// 行列シートのインスタンスを構成し、名前を付けたシートを付加

MatrixLayer matLy1(strFullName);



// 名前付きシートに既存行列のインスタンスを付加

matLy2.Attach(strFullName);
```

2. 行列ブックには、行列レイヤのコレクションが含まれています。全ての特定の行列ブック内の行列レイヤを 、**foreach** ステートメントを使用してループします。

```
MatrixPage matPage("MBook1");

foreach(Layer ly in matPage.Layers)

    out_str(ly.GetName());
```

3. 名前かインデックスで特定の行列シートにアクセス

```
// ページ MBook1 には、最低でも 2 つの行列シートがあると仮定

// それらのシート名は、MSheet1 と MSheet2

MatrixPage matPage("MBook1");

MatrixLayer lyFirst = matPage.Layers(0); // インデックスによる

MatrixLayer lySecond = matPage.Layers("MSheet2"); //名前による
```

行列シートのプロパティを修正する

次数の取得と設定

Origin では、行列シート内のすべての行列オブジェクトは同じ次数(列と行の数が同じ)を共有します。

1. 行列シートの行と列の数を取得するには、まず行列シートの最初の行列オブジェクトを取得し、MatrixObject クラスのメソッド (GetNumCols と GetNumRows) を使用します。

```
// 行と列の数を取得

MatrixLayer ml = Project.ActiveLayer(); // アクティブ行列シートを取得

MatrixObject mo = ml.MatrixObjects(0); // 第 1 行列オブジェクトを取得



int nNumRows = mo.GetNumRows();  // 行数を取得

int nNumCols = mo.GetNumCols();  // 列数を取得
```

2. 行列シートの次数を設定するには、MatrixLayer::SetSize メソッドを使用します。

```
// 行と列の数を設定

MatrixLayer ml = Project.ActiveLayer(); // アクティブ行列シートを取得

ml.SetSize(-1, 5, 5);  // 次数を 5x5 に設定
```

3. MatrixObject クラスは、次数設定のためのメソッド SetSize を提供します。しかし、同じ行列シート内の全行列オブジェクトの次数は同じであるため、MatrixObject で定義されるメソッドであるにも関わらず、行列シートの次数が変更されることに注意してください。

```
// 行と列の数を設定

MatrixLayer ml = Project.ActiveLayer(); // アクティブな行列シートを取得

MatrixObject mo = ml.MatrixObjects(0); // 第 1 オブジェクトを取得



int nNumRows = 5, nNumCols = 5;

mo.SetSize(nNumRows, nNumCols);  // 次数を 5x5 に設定
```

4. 行列は、番号が付けられた列と行を持ち、これらは等間隔で線形にマッピングされた X 値および Y 値です。SetXY メソッドで XY のマッピング座標を設定できます。Note: このメソッドは、行列オブジェクトによって利用可能ですが、XY マッピングは同じ行列シート上で共有します。

```
MatrixLayer ml = Project.ActiveLayer();  // アクティブレイヤを取得

MatrixObject mo = ml.MatrixObjects(0);  // 第 1 行列オブジェクトを取得

mo.SetXY(-10, 20, -2.3, 12.4);  // X を -10 から 20 にし、Y を -2.3 から 12.4 に設定
```

ラベルの取得と設定

行列のラベルには、X、Y、Z に対するロングネーム、単位、コメントが含まれます。X と Y のラベルは、行列シート内の全行列オブジェクトで共通で、Z のラベルは各行列オブジェクトで固有です。次のコードは、ラベルの取得と設定の方法を示しています。

1. XY ラベルを設定

```
MatrixPage mp("MBook1");

MatrixLayer ml = mp.Layers(0); // 第 1 行列シート



Tree tr;

tr.Root.Dimensions.X.LongName.strVal = "X Values";

tr.Root.Dimensions.X.Unit.strVal = "X Units";

tr.Root.Dimensions.X.Comment.strVal = "X Comment";



tr.Root.Dimensions.Y.LongName.strVal = "Y Values";

tr.Root.Dimensions.Y.Unit.strVal = "Y Units";

tr.Root.Dimensions.Y.Comment.strVal = "Y Comment";



// XY ラベルのための行列シート上でフォーマットをセット

if( 0 == ml.UpdateThemeIDs(tr.Root) )

        ml.ApplyFormat(tr, true, true);
```

2. XY ラベルを取得

```
MatrixPage mp("MBook1");

MatrixLayer ml = mp.Layers(0); // 第 1 行列シート



// 行列オブジェクトではなく行列シートの XY ラベルを取得

Tree tr;

tr = ml.GetFormat(FPB_ALL, FOB_ALL, TRUE, TRUE);



TreeNode trX = tr.Root.Dimensions.X;

if( !trX.LongName.IsEmpty() )

        printf("X Long Name:%s¥n", trX.LongName.strVal);

if( !trX.Unit.IsEmpty() )

        printf("X Unit:%s¥n", trX.Unit.strVal);

if( !trX.Comment.IsEmpty() )

        printf("X Comment:%s¥n¥n", trX.Comment.strVal);



TreeNode trY = tr.Root.Dimensions.Y;

if( !trY.LongName.IsEmpty() )

        printf("Y Long Name:%s¥n", trY.LongName.strVal);

if( !trY.Unit.IsEmpty() )

        printf("Y Unit:%s¥n", trY.Unit.strVal);

if( !trY.Comment.IsEmpty() )

        printf("Y Comment:%s¥n", trY.Comment.strVal);
```

3. Z ラベルを設定

```
MatrixPage mp("MBook1");

MatrixLayer ml = mp.Layers(0); // 第 1 行列シート

MatrixObject mo = ml.MatrixObjects(0);// 第 1 行列オブジェクト
```

```
// フォーマットツリーを構成して文字列値をツリーノードに割り当て
Tree tr;
tr.Root.LongName.strVal = "Z Long Name";
tr.Root.Unit.strVal = "Z Units";
tr.Root.Comment.strVal = "Z Comment";

// 行列シートではなく、行列オブジェクトに Z ラベルを適用
if( 0 == mo.UpdateThemeIDs(tr.Root) ) // 各ツリーノードの ID を追加
        mo.ApplyFormat(tr, true, true);        // 適用
```

4. Z ラベル取得

```
MatrixPage mp("MBook1");
MatrixLayer ml = mp.Layers(0); // 第 1 行列シート
MatrixObject mo = ml.MatrixObjects(0);

Tree tr;
tr = mo.GetFormat(FPB_ALL, FOB_ALL, TRUE, TRUE);

printf("Z Short Name:%s¥n", tr.Root.ShortName.strVal);
if( !tr.Root.LongName.IsEmpty() )// 空でない場合
        printf("Z Long Name is %s¥n", tr.Root.LongName.strVal);
if( !tr.Root.Unit.IsEmpty() )
        printf("Z Unit is %s¥n", tr.Root.Unit.strVal);
if( !tr.Root.Comment.IsEmpty() )
        printf("Z Comment is %s¥n", tr.Root.Comment.strVal);
```

行列シートのフォーマット

テーマツリーを使用して、プログラムで行列シートをフォーマットできます。

次のサンプルは、アクティブ行列シートないのセルブロックをフォーマットし、背景を青色、テキストを明るい深紅色にします。

```
MatrixLayer ml = Project.ActiveLayer();



Tree tr;

tr.Root.CommonStyle.Fill.FillColor.nVal = SYSCOLOR_BLUE;

tr.Root.CommonStyle.Color.nVal = SYSCOLOR_LTMAGENTA;



DataRange dr;

dr.Add(NULL, ml, 2, 2, 5, 3); // 第 1 行、列、最後の行、列

if( 0 == dr.UpdateThemeIDs(tr.Root) )

    dr.ApplyFormat(tr, TRUE, TRUE);
```

行列セルテキスト色の取得と設定

次のサンプルでは、せるのテキスト色を取得して設定します。

```
// 単純なユーティリティ関数に 'set' コードを内包

bool setCellTextColor(Datasheet& ds, int row, int col, uint color)

{

    Grid grid;

    if( !grid.Attach(ds) )

        return false;

    vector<uint> vTextColor(1);

    vTextColor[0] = color;

    return grid.SetCellTextColors(vTextColor, col, row, row);

}



// シンプルなユーティリティ関数に 'get' コードを内包

bool getCellTextColor(Datasheet& ds, int row, int col, uint& color)
```

```
{
    Grid grid;
    if( !grid.Attach(ds) )
        return false;
    vector<uint> vTextColor;
    if( !grid.GetCellTextColors(vTextColor, col, row, row) )
        return false;
    color = vTextColor[0];
    return true;
}

// 上述のユーティリティ関数をテストするためのシンプルな関数
void testCellTextColor(int nRow = 3, int nCol = 4)
{
    MatrixLayer ml = Project.ActiveLayer();
        // nRow, nCol は LT/GUI インデックス, 1-offset を使用するが OC は 0-offset
        int row = nRow-1, col = nCol-1;
    setCellTextColor(ml, row, col, SYSCOLOR_BLUE);

    uint color;
    getCellTextColor(ml, row, col, color);
    printf("color == %d¥n", color);
}
```

6.2.2 行列シートのデータ操作

行列シートと行列オブジェクト間の変換

Origin で行列シートは複数の行列オブジェクトを持つことができます。関数 matobj_move を使用して、複数の行列オブジェクトを分解して行列シートに分けルことができます。あるいは、複数の行列シートを一つに統合するができます。どちらの場合も次数は同じです。

```
// このサンプルは、3 つのシートの行列オブジェクトを統合するためのコードの一部

// 新しいシート

MatrixPage mp("MBook1");  // 行列ブック

MatrixLayer ml1 = mp.Layers(1);  // 2 番目のシート

MatrixLayer ml2 = mp.Layers(2);  // 3 番目のシート

MatrixLayer ml3 = mp.Layers(3);  // 4 番目のシート



MatrixLayer mlMerge;

mlMerge.Create("Origin");  // 統合のたの新しいシート

MatrixObject mo1 = ml1.MatrixObjects(0);  // 2 番目のシート内の行列オブジェクト

MatrixObject mo2 = ml2.MatrixObjects(0);  // 3 番目のシート内の行列オブジェクト

MatrixObject mo3 = ml3.MatrixObjects(0);  // 4 番目のシート内の行列オブジェクト

matobj_move(mo1, mlMerge);  // 行列オブジェクトをシートの最後に移動

matobj_move(mo2, mlMerge);

matobj_move(mo3, mlMerge);
```

6.3 行列オブジェクト

行列オブジェクト(MatrixObject クラス)は強力な行列データの基本的な単位で、これの入れものとして行列シートがあります。この 2 つの関係はワークシート内の列のようなものです。次のページでは行列オブジェクトの操作に関する実践的なサンプルを紹介します。

6.3.1 行列オブジェクトの基本操作

行列シートは複数の行列オブジェクトを持つことができ、それらは共通の次数を持ちます。行列オブジェクトはワークシートと同じように認識することができ、追加や削除ができます。以下のセクションは行列オブジェクトの基本的な操作について、実践的なサンプルを通して紹介します。

行列オブジェクトの追加または挿入

MatrixLayer::SetSize を使用すると行列シート内に複数の行列オブジェクトをセットして行列オブジェクトを追加する事ができます。

```
// アクティブ行列シートに 5 つの行列オブジェクトをセット

MatrixLayer ml = Project.ActiveLayer();

ml.SetSize(5);
```

MatrixLayer::Insert メソッドは、現在の行列オブジェクトの前に、指定された行列オブジェクトの番号を挿入します。

```
// 行列オブジェクトをシートに追加

MatrixLayer ml = Project.ActiveLayer(); // アクティブ行列シートを取得



int nNum = 1; // 追加された行列オブジェクトの番号

int nPos = -1; // -1, 最後に追加

int nDataType = -1; // 任意, -1 はデフォルトで double 型

int index = ml.Insert(nNum, nPos, nDataType); // 最初のものもインデックスを返す
```

行列オブジェクトをアクティブ化する

行列シートの中にある行列オブジェクトをアクティブ化するには、MatrixLayer::SetActive を使用します。

```
MatrixLayer ml = Project.ActiveLayer();

ml.SetActive(2);  // 第 3 行列オブジェクトをアクティブ化(インデックスは 0 ベース)
```

行列オブジェクトにアクセスする

行列オブジェクトにアクセスするには、MatrixLayer からの MatrixObjects のコレクションを使用します。

```
// 名前で 1 つの行列オブジェクトに付加

MatrixPage matPage("MBook3");



// 行列ページからシート名 MSheet1 を付加

// インデックスで行列ページからのシート取得もサポート

MatrixLayer ml1 = matPage.Layers("MSheet1");



// インデックスによりシートから行列オブジェクトを取得

MatrixObject mo = ml1.MatrixObjects(0);



// 行列オブジェクトのデータ型は、行列ウィンドウ内で一致している必要がある

if( FSI_SHORT == mo.GetInternalDataType() )

{

        matrix<short>& mat = mo.GetDataObject();

}
```

行列オブジェクト削除する

行列シートから、指定した数の行列オブジェクトを削除するには、MatrixLayer::Delete メソッドを使用します。

```
// シートから行列オブジェクトを削除

MatrixLayer ml = Project.ActiveLayer(); // アクティブな行列シートを取得



// はじめから 2 つの行列オブジェクトを削除

int nPos = 0;

int nNum = 2;

ml.Delete(nPos,    nNum);
```

イメージモードとデータモードを切り替える

MatrixLayer::SetViewImage メソッドは、インデックスで指定した行列の、イメージモードとデータモードの切り替えオプションを提供します。

```
// イメージモードにセット

MatrixLayer ml = Project.ActiveLayer(); // アクティブ行列シートを取得



int nImgIndex = 0;

MatrixObject mo = ml.MatrixObjects(nImgIndex);



if( !mo.IsImageView() )

{

        BOOL bAllObjs = FALSE;

        ml.SetViewImage(TRUE, bAllObjs, nImgIndex);  // データモードは FALSE

}
```

ラベルの取得と設定

各行列オブジェクトで、ロングネーム、コメント、単位をセットできます。Z ラベルの設定については、行列シートの基本操作の章の、Z ラベルの取得と設定をご覧ください。

データ型とフォーマット

データ型の取得と設定

行列オブジェクトの内部データ型には、double, real, short, long, char, text, mixed, byte, ushort, ulong, complex 型などがあります。Origin C は、MatrixObject クラス内に GetInternalDataType と SetInternalDataType メソッドを提供し、行列オブジェクトの内部データタイプの取得と設定に使われます。

```
// データ型の取得と設定

MatrixLayer ml = Project.ActiveLayer(); // アクティブ行列シートを取得

MatrixObject mo = ml.MatrixObjects(0);



if( mo.GetInternalDataType() != FSI_BYTE ) // データ型を取得

{

        // OCD_RESTORE はデータをバックアップし

        // 型を変更した後元に戻す

        DWORD dwFlags = OCD_RESTORE;

        mo.SetInternalDataType(FSI_BYTE, dwFlags); // データ型をセット

}
```

フォーマットの取得と設定

MatrixObject::GetFormat と MatrixObject::SetFormat は、行列オブジェクトのデータフォーマットを取得し、設定することができます。

```
// データフォーマットを取得して設定

MatrixLayer ml = Project.ActiveLayer(); // アクティブ行列シートを取得

MatrixObject mo = ml.MatrixObjects(0);



int nFormat = mo.GetFormat(); // OKCOLTYPE_NUMERIC( = 0) のみサポートされる

mo.SetFormat(OKCOLTYPE_NUMERIC);
```

6.3.2　行列オブジェクトのデータ操作

式で値を設定

DataObject::SetFormula と **DataObject::ExecuteFormula** メソッドは、列/行列の値をセットするのに使用され、これは、**値の設定**ダイアログによる操作と同様です。以下のサンプルでは、式を使用して行列オブジェクトに値を設定する方法を示します。

```
// 新しい行列ウィンドウ

MatrixPage matPage;

matPage.Create("Origin");

MatrixLayer ml = matPage.Layers(); // アクティブ行列シートを取得



// 式を設定して実行

MatrixObject mo = ml.MatrixObjects(0); //最初の matrixobject を取得

mo.SetFormula("sin(i) + cos(j)");

mo.ExecuteFormula();
```

行列データのコピー

matobj_copy 関数は、行列データを コピーする時に使用します。

```
MatrixLayer mlSrc = Project.ActiveLayer();  // アクティブ行列シートを取得

MatrixObject moSrc = mlSrc.MatrixObjects(0);  // シート内の第 1 行列オブジェクトを取得

MatrixLayer mlDst;

mlDst.Create("Origin");  // 新しい行列シートを取得

MatrixObject moDst = mlDst.MatrixObjects(0);  // 第 1 行列オブジェクトを取得

bool bRet = matobj_copy(moDst, moSrc);  // アクティブデータを新しく作成した行列にコピー
```

行列データの演算操作

行列上で演算操作を実行するには、行列オブジェクトからデータ行列にデータを取得し、計算して、結果を行列オブジェクトに戻します。行列の乗算操作には、constant, dot multiply, dot divide, dot power, cross, cumulative product, cumulative sum, difference, などがあります。

次の 2 つの行列に対するサンプルでは、定数による乗算と、行列同士の乗算をの方法を紹介します。

定数による行列の掛け算

```
MatrixLayer ml = Project.ActiveLayer(); // アクティブ行列シートを取得

MatrixObject mo = ml.MatrixObjects(0); // 第 1 行列オブジェクトを取得



//行列データの内部データオブジェクトのリファレンスを取得

// 行列のデータ型を double に割り当て

matrix<double>& mat = mo.GetDataObject();
```

```
// 行列内の各データに 10 を掛ける。ウィンドウにこの変更を反映

mat = mat * 10;
```

2 つの行列の掛け算

```
// 2 つの行列ページを付加

MatrixPage matPage1("MBook1");

MatrixPage matPage2("MBook2");

if( !matPage1 || !matPage2 )

        return;



// 名前かインデックスでページから行列シートを取得

MatrixLayer matLayer1 = matPage1.Layers("MSheet1");

MatrixLayer matLayer2 = matPage2.Layers(1); // 第 2 シートを取得

if( !matLayer1 || !matLayer2 )

        return;



// インデックで行列シートから行列オブジェクトを取得。名前は不可。

MatrixObject mo1 = matLayer1.MatrixObjects(0);

MatrixObject mo2 = matLayer2.MatrixObjects(0);



// 行列ウィンドウの内部データオブジェクトをのリファレンスを取得

matrix<double>& mat1 = mo1.GetDataObject();

matrix<double>& mat2 = mo2.GetDataObject();



// 新しい行列ウィンドウを用意

MatrixPage matPageNew;

matPageNew.Create("Origin");

MatrixLayer mlNew = matPageNew.Layers(0);

MatrixObject moNew = mlNew.MatrixObjects(0);
```

```
matrix<double>& matNew = moNew.GetDataObject();



// mat1 から新しい行列に値をコピー

matNew = mat1;



// 各行列の値を掛け算して結果を

// 新しい行列ウィンドウに出力

matNew.DotMultiply(mat2);
```

行列オブジェクトとベクターを変換する

メソッド matrixbase::GetAsVector と matrixbase::SetByVector は、行列オブジェクトとベクターを変換するとに使用できます。

```
// ベクターに

MatrixLayer ml = Project.ActiveLayer();  // アクティブ行列シート

MatrixObject mo = ml.MatrixObjects(0);  // 第 1 行列オブジェクト

matrixbase &mb = mo.GetDataObject();  // 行列オブジェクトからデータを取得

vector vb;

mb.GetAsVector(vb);  // 行列データをベクターに変換



// ベクターから

MatrixLayer ml1;

ml1.Create("Origin");  // 行列シートを作成

MatrixObject mo1 = ml1.MatrixObjects(0);  // 行列オブジェクトを取得

matrixbase &mb1 = mo1.GetDataObject();  // データオブジェクトを取得

mb1.SetSize(2, 3);  // 2 行*3 列のサイズにセット

vector v = {1, 2, 3, 4, 5, 6};  // Vector data

// 行列オブジェクトにベクターをセット

// 第 1 行:1, 2, 3

// 第 2 行:4, 5, 6

int iRet = mb1.SetByVector(v);
```

複素数値を持つ行列を操作する

Origin C では、matrixbase クラス内の making a complex matrix from two real matrices, getting real ,imaginary, getting phase , amplitude, calculating conjugate などを含むメソッドのセットは複素数を扱うために使用します。

次のコードは、2 つの実数の行列データで複素数行列として行列を設定するために使用されます。そして、その実数、虚数、位相、振幅を異なる行列オブジェクトに分け、元の行列オブジェクトを削除するために Conjugate を使用します。

```
void MatrixObject_Complex_EX()

{

        // 実数の元データ

        matrix mR =

        {

                {2, 2, 2, 0},

                {0, 1, 99, 99}

        };

        // 虚数の元データ

        matrix mI =

        {

                {3, -3, 0, 3},

                {0, 99, 1, 99}

        };

        matrix<complex> mC;

        // 複素数データを作成

        int iRet = mC.MakeComplex(mR, mI);

        if(iRet == 0)

        {

                // 複素数データ用の行列シートを作成

                MatrixLayer ml;

                ml.Create("Origin");

                MatrixObject mo = ml.MatrixObjects(0);

                ml.SetInternalData(FSI_COMPLEX);
```

```
matrixbase &mb = mo.GetDataObject();
mb = mC;

// 実部を取得
matrix mReal;
mb.GetReal(mReal);
// 虚部を取得
matrix mImg;
mb.GetImaginary(mImg);
// 位相を取得
matrix mPha;
mb.GetPhase(mPha);
// 振幅を取得
matrix mAmp;
mb.GetAmplitude(mAmp);
// 結果のために新しい行列シートを作成
MatrixLayer mlRes;
mlRes.Create("Origin");
// この行列と同じサイズの 4 つの行列オブジェクトを作成
mlRes.SetSize(4, mb.GetNumRows(), mb.GetNumCols());
MatrixObject moReal = mlRes.MatrixObjects(0);
MatrixObject moImg = mlRes.MatrixObjects(1);
MatrixObject moPha = mlRes.MatrixObjects(2);
MatrixObject moAmp = mlRes.MatrixObjects(3);
matrixbase &mbReal = moReal.GetDataObject();
matrixbase &mbImg = moImg.GetDataObject();
matrixbase &mbPha = moPha.GetDataObject();
matrixbase &mbAmp = moAmp.GetDataObject();
```

```
                mbReal = mReal;  // 行列オブジェクトに実部をセット

                mbImg = mImg;  // 行列オブジェクトに虚部をセット

                mbPha = mPha;  // 行列オブジェクトに位相をセット

                mbAmp = mAmp;  // 行列オブジェクトに振幅をセット



                // Conjugate を使用して元の複素数行列を削除

                mb.Conjugate();

        }

}
```

行列オブジェクトデータを変換する

Origin C には、行列変換のために行列内のメソッドのセットを含みます。例えば、flip a matrix horizontally や vertically, rotate a matrix, shrink a matrix, transpose a matrix, などです。

```
MatrixLayer ml = Project.ActiveLayer();

MatrixObject mo = ml.MatrixObjects(0);

matrixbase &mb = mo.GetDataObject();



mb.FlipHorizontal();  // 平行に移動

mb.FlipVertical();  // 垂直に移動

mb.Rotate(90);  // 90 度時計回りに回転

mb.Shrink(2, 2);  // 行と列を因子 2 で縮小する

mb.Transpose();  // 転置
```

6.3.3　行列からワークシートに変換する

分析または、グラフ作成時に、行列からワークシートに変換またはその反対の操作により、データを再構成する必要があるかもしれません。このページでは、行列をワークシートに変換する時のサンプルと情報を紹介します。なお、反対の操作は、ワークシートを行列に変換するを確認してください。

行列からワークシート

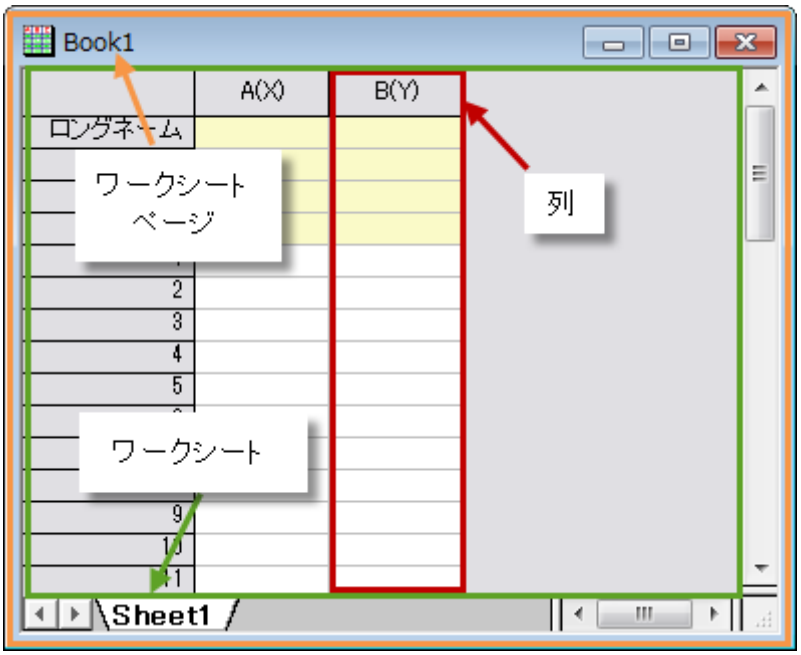
行列オブジェクトデータをワークシートに変換するには、まず行列オブジェクト内のデータをデータ行列に取得し、クラス内に定義された CopyTo メソッドを使用します。

ここでは、すべての行列オブジェクトを直接ワークシートに変換する方法を示します。

```
// アクティブ行列オブジェクトを新しく作成した行列に直接変換

// 転置なし、列タイプは行列と同じ

MatrixLayer ml = Project.ActiveLayer();  // 行列シートをアクティブにする

MatrixObject mo = ml.MatrixObjects(0);  // 第 1 行列オブジェクトを取得

matrixbase &mb = mo.GetDataObject();  // 行列オブジェクトからデータを取得

Worksheet wks;

wks.Create("Origin");  // 新しいワークシートを作成

mb.CopyTo(wks, 0, 0, -1, -1, 0, 0, FALSE, TRUE);  // データをワークシートに変換
```

7 ワークブック、ワークシート、ワークシート列

Origin C の WorksheetPage クラスは Origin のワークブックと操作するときに使用します。各ワークブックは、Worksheets のコレクションを含み、各行列レイヤは、Columns のコレクションを含みます。



7.1 ワークブック

Origin C の WorksheetPage クラスは、Origin のワークブックに共通するメソッドとプロパティを提供します。このクラスは、Page クラス由来のもので、そのメソッドとプロパティを継承します。

7.1.1 ワークブックの基本操作

新しいワークブックを作成

Create メソッドは、新しいワークブックの作成に使用されます。

```
// STAT テンプレートを使用して非表示のワークブックを作成

WorksheetPage wksPg;

wksPg.Create("STAT", CREATE_HIDDEN);
```

ワークブックを開く

ワークブックのデータが拡張子 ogw のファイルとして保存されている場合、Open メソッドで開く事ができます。

```
Worksheet wks;  // ワークシートに属した Open メソッド

string strOGW = "D:¥¥Book1.ogw";  // ワークブックのパス

wks.Open(strOGW);  // ワークブックを開く
```

ワークブックにアクセスする

既存のワークブックにアクセスするには、複数の方法があります。Project クラスは、プロジェクト内の全ワークブックのコレクションを含みます。次のサンプルでは、それらをどのようにループするかを示します。

```
foreach(WorksheetPage wksPg in Project.WorksheetPages)

    out_str(wksPg.GetName()); // ワークブック名を出力
```

また、Collection クラスの Item メソッドに、ワークブックのインデックスを渡すことによってワークブックにアクセスできます。

```
WorksheetPage wksPg;

wksPg = Project.WorksheetPages.Item(2);

if(wksPg) // 3 番目のワークブックがある場合

    out_str(wksPg.GetName()); // ワークブック名を出力
```

ワークブック名がわかっている場合、クラスコンストラクタにその名前を渡すことで、ワークブックにアクセスできます。

```
WorksheetPage wksPg("Book1");

if(wksPg) // ワークブック名が"Book1"のワークブックがある場合

    wksPg.SetName("MyBook1"); // ワークブックの名前を変更
```

ワークブックを保存する

Origin はワークブックのデータをファイル(拡張子 .ogw)として保存、データなしのテンプレートで保存(拡張子 .otw)、分析を行ったワークブックに関しては分析テンプレート(拡張子 .ogw)として保存できます。メソッド SaveToFile と SaveTemplate は、ワークブックを *.ogw と *.otw ファイルとして保存するのに使用されます。

```
WorksheetPage wksPg("Book1");

// ワークブックを OGW ファイルとして保存

bool bRet1 = wksPg.SaveToFile("D:¥¥"+ wksPg.GetName() + ".ogw");

// ワークブックを OTW テンプレートとして保存

bool bRet2 = wksPg.SaveTemplate("D:¥¥"+ wksPg.GetName() + ".otw");
```

ワークブックの表示と非表示

WorksheetPage クラスは、OriginObject クラスから Show プロパティを継承し、自身の表示/非表示を決めます。

```
WorksheetPage wksPg("Book1");

wksPg.Show = false;  // ワークブックを非表示にする。true の場合、ワークブックを表示。
```

ワークブックをアクティブにする

ワークブックをアクティブにするために、PAGE_ACTIVATE パラメータ値を渡すことによって、SetShow メソッドを使用できます。

```
WorksheetPage wksPg("Book1");

wksPg.SetShow(PAGE_ACTIVATE);  // Activate the workbook

// 異なる値を渡すことで多くの操作を完了できます。たとえば、

// wksPg.SetShow(PAGE_HIDDEN);  // ワークブック非表示

// wksPg.SetShow(PAGE_MINIMIZED);  // ワークブック最小化

// wksPg.SetShow(PAGE_MAXIMIZED);  // ワークブック最大化
```

ワークブックを削除する

すべての Origin C の内部クラスは、OriginObject クラスから派生したものです。このクラスは、オブジェクトを破棄するのに使用する Destroy メソッドがあります。ワークブックまたは行列ブックでこのメソッドを呼び出すと、ワークブックの中にあるすべてのシート、各シート内のすべての列または行列オブジェクトを削除します。

```
WorksheetPage wksPg;

wksPg = Project.WorksheetPages.Item(0); // プロジェクト内の最初のワークブックを取得

if( wksPg ) //  ワークブックがあれば

    wksPg.Destroy(); // ワークブックを削除
```

ワークブックを複製する

WorksheetPage クラス(ワークブックに対して)は、Page クラスから派生したものです。このクラスは、元のページを複製するのに使用する Clone メソッドがあります。

```
// データとスタイルと一緒に "Book1"を複製

// 呼び出す前にウィンドウが存在することを確認

WorksheetPage wksPage("Book1");

WorksheetPage wksPage1 = wksPage.Clone();
```

ワークブックに名前とラベルを付ける

ワークブックにはショートネーム、ロングネーム、コメントがあります。OriginObject クラスで定義された継承メソッド SetName, SetLongName, SetComments は、ワークブック名(ショートネーム、ロングネームとも)とコメントを制御するために使用できます。

```
WorksheetPage wksPg("Book1");

if(wksPg)

{

        wksPg.SetName("MyBook");  // ワークブック名変更

        wksPg.SetLongName("This is Long Name", false);  // ロングネームをセット

        wksPg.SetComments("Comments");  // コメントをセット

}
```

また、Label プロパティもロングネーム変更に使用できます。TitleShow プロパティじゃワークブックタイトルにロングネーム、ショートネームをどのように表示するか制御します。

```
WorksheetPage wksPg1("Book2");

if(wksPg1)

{

        wksPg1.Label = "My Label";  // ラベル(ロングネームともいう)をセット

        // ワークブックタイトルにはラベルのみ表示

        wksPg1.TitleShow = WIN_TITLE_SHOW_LABEL;

        // ワークブックタイトルにはショートネームのみ表示

        // wksPg1.TitleShow = WIN_TITLE_SHOW_NAME;

        // ワークブックタイトルにショートネームとラベル両方表示

        // wksPg1.TitleShow = WIN_TITLE_SHOW_BOTH;

}
```

7.1.2　ワークブックの操作

Origin は分割、統合、複製など、Origin C を使用してブックを操作するための機能を提供しています。

ワークブックを統合

複数のワークブックを統合して一つにまとめる場合、実際には、ソースワークシートからターゲットワークブックにコピーします。ワークブックにワークシートを追加するには、AddLayer メソッドを使用できます。

次のサンプルは、現フォルダ内のすべてのワークブックを統合して新しいワークブックを作成します。

```
WorksheetPage wksPgTarget;

wksPgTarget.Create("Origin");  // 目的のワークブックを作成

Folder fld = Project.ActiveFolder();  // アクティブ/現在のフォルダを取得
```

```
foreach(PageBase pb in fld.Pages)

{  // フォルダ内の前ページでループ

        WorksheetPage wksPgSource = pb;  // Page から WorksheetPage に変換

        // 変換失敗の場合

        if(!wksPgSource)

        {

                continue;  // 次のページ

        }

        // ターゲットワークブックをスキップ

        if(wksPgTarget.GetName() == wksPgSource.GetName())

        {

                continue;

        }

        // ワークブック内のすべてのワークシートで統合をループ

        foreach(Layer lay in wksPgSource.Layers)

        {

                Worksheet wks = lay;  // ワークシート取得

                // 目的のワークブックにワークシートを追加

                wksPgTarget.AddLayer(wks, 0, false);

        }

        // ソースワークブックを保持しない場合破棄

        wksPgSource.Destroy();

}
```

ワークブックの分割

上記のサンプルでは複数のワークブックを統合して一つにまとめましたが、ワークブックを分割し、単一ワークシートを持つ複数のブックにすることができます。

```
WorksheetPage wksPgSource("Book1");  // 複数シートをもつワークブック

// 全てのワークシートでループ

foreach(Layer lay in wksPgSource.Layers)
```

```
{

        Worksheet wks = lay;  // ワークシートを取得

        WorksheetPage wksPgTarget;

        wksPgTarget.Create("Origin");  // 新しいワークブックを作成

        wksPgTarget.AddLayer(wks);  // 新しいワークブックのためにワークシートを追加

        wksPgTarget.Layers(0).Destroy();  // 最初のワークシートを削除

}
```

7.2 ワークシート列

7.2.1 ワークシート列

Origin C は、ワークシート列を取り扱う Column クラスを提供しています。Column オブジェクトは、通常、列に含まれるデータセットのスタイル、フォーマット、データ型を制御するのに使用します。Column クラスを使用する方法を示すサンプルコードが次のセクションで提供されています。

7.2.2 ワークシート列操作

ワークシート列の操作を行うには、Column クラスか Worksheet クラスを使用します。

列の追加・挿入

ワークシートの最後に新しい列を追加する場合、Worksheet クラスの AddCol メソッドを使用し、指定した位置の直前に列を挿入するときは InsertCol を使用します。

```
// デフォルト名で列を追加

int nColIndex = wks.AddCol();



// strName という名前の列を追加;

int nColIndex = wks.AddCol("AA", strName); // 列のインデックスを返す

// AA という名前の列が既にある場合、自動で名付け

out_str(strName);

Column col(wks, nColIndex); // 列インデックスで列オブジェクトを構築




// 新しい列を 1 列目として挿入

int nPos = 0; // 挿入する位置
```

```
string strNewCreated; // 新しい列の名前

// MyCol という列名が既にある場合は、自動で名付け

if( wks.InsertCol(nPos, "MyCol", strNewCreated) )

{

        printf("Insert column successfully, name is %s¥n", strNewCreated);

}
```

列の削除

Worksheet::DeleteCol メソッドはワークシートから列を削除する事ができます。

```
// インデックスで列を削除

wks.DeleteCol(0);
```

列の名前を変更する・ラベルを付ける

列の名前(ショートネーム)を変更するには、SetName メソッドを使用します。

```
Column col = wks.Columns(0);  // ワークシート内の１列目を取得

BOOL bRet = col.SetName("MyNewName");  // 列の名前を変更
```

ワークシート列ラベルには、ロングネーム、単位、コメント、パラメータ、ユーザ定義ラベル等がサポートされています。
Origin C コードでラベルの表示/非表示や、特定の列ラベルにテキストを追加できます。

```
Worksheet wks;

wks.Create();



Grid gg;

gg.Attach(wks);



// パラメータラベルが非表示の場合、表示する

bool bShow = gg.IsLabelsShown(RCLT_PARAM);

if( !bShow )

        gg.ShowLabels(RCLT_PARAM);



wks.Columns(0).SetLongName("X Data");

wks.Columns(1).SetLongName("Y Data");
```

```
wks.Columns(0).SetComments("This is a test");


wks.Columns(0).SetUnits("AA");
wks.Columns(1).SetUnits("BB");


// 2 列のパラメータラベルにテキストを入力
wks.Columns(0).SetExtendedLabel("Param A", RCLT_PARAM);
wks.Columns(1).SetExtendedLabel("Param B", RCLT_PARAM);
```

RCLT_PARAM は、パラメータ列ラベルタイプです。他のタイプについては、*OriginC\system\oc_const.h* の ROWCOLLABELTYPE enum を確認してください。

列の表示 / 非表示

列を表示/非表示にするには、Workhseet::ShowCol メソッドを使用します。

```
wks.ShowCol(1, 1, false); // 列 1 を非表示にする
```

列の移動と交換

列を移動したり交換する場合、Worksheet クラス、Datasheet クラスが MoveColumns メソッドと[[OriginC:Datasheet-SwapColumns|SwapColumns]を提供しています。

```
// 3 列移動 - 列 5 を最初の列へ
// このサンプルでは最低限 7 列あるワークシートが必要
Worksheet wks = Project.ActiveLayer();
if(wks)
    wks.MoveColumns(4, 3, MOVE_COL_TO_FIRST);


// アクティブワークシートの列の順番を逆に
for(int ii = 1; ii <= wks.GetNumCols() / 2 ; ii++)
    wks.SwapColumns(ii - 1, wks.GetNumCols() - ii);
```

列にスパークラインを追加する

列にスパークラインを追加するために、Origin C では RCLT_SPARKLINE ラベルタイプの wks_set_show_labels が提供されています。

```
// サンプリング間隔とスパークラインを表示するためにアクティブワークシートを構成

// 現在のラベルに追加

Worksheet wks = Project.ActiveLayer();

vector<int> vn = {RCLT_SAMPLE_RATE, RCLT_SPARKLINE};

wks_add_show_labels(wks, vn, false);
```

属性、フォーマット、サブフォーマット

属性の取得と設定

```
Worksheet wks = Project.ActiveLayer();

Column col(wks, 0);



// 列タイプを取得

// 0:Y

// 1:なし

// 2:Y エラー

// 3:X

// 4:L

// 5:Z

// 6:X Error

int nType = col.GetType();

out_int("Type:", nType);



// 列属性をセット See more define OKDATAOBJ_DESIGNATION_* in oc_const.h

col.SetType(OKDATAOBJ_DESIGNATION_Z);
```

データフォーマットの取得と設定

```
// データフォーマットを取得して設定

// 列のデフォルトフォーマットは、 OKCOLTYPE_TEXT_NUMERIC

// 列のフォーマットを日付に変更

if( OKCOLTYPE_DATE != col.GetFormat() )

{

        col.SetFormat(OKCOLTYPE_DATE);

}
```

データサブフォーマットの取得と設定

```
// データサブフォーマットの取得と設定

// サブフォーマットのオプションは、フォーマットのタイプ

// 数値、日付、時間などに応じて異なる

if( LDF_YYMMDD != col.GetSubFormat() )

{

        col.SetSubFormat(LDF_YYMMDD);

}
```

7.2.3　ワークシート列データの操作

基本的な算術操作

列データに対して基本的な算術演算の操作を行うには、列データをベクトルに取得し、対応ベクトルで操作します。

```
// 1、2 列目からデータを取得

// この 2 列を足す

// そして結果を 3 列目に出力

Worksheet wks = Project.ActiveLayer();

if(!wks)

{

        return;

}

Column col1 = wks.Columns(0);  // 1 列目
```

```
Column col2 = wks.Columns(1);  // 2 列目

Column col3 = wks.Columns(2);  // 3 列目

vectorbase &v1 = col1.GetDataObject();  // データオブジェクトを取得

vectorbase &v2 = col2.GetDataObject();

vectorbase &v3 = col3.GetDataObject();

v3 = v1 + v2;  // 足し合わす
```

式で値を設定

DataObject::SetFormula と DataObject::ExecuteFormula メソッドは、**値の設定**ダイアログのように、列/行列値の設定に使用できます。次のサンプルでは、3 列のワークシートを作成し、各列の値を式で設定します。

```
Worksheet wks;

wks.Create("origin", CREATE_VISIBLE);

wks.AddCol();



// 1 列目の値をセット

Column colA;

colA.Attach(wks, 0);

colA.SetFormula("5*(i-1)");

colA.ExecuteFormula();



// 次の 2 列は再計算を自動にセット

Column colB;

colB.Attach(wks, 1);

colB.SetFormula("sin(4*col(A)*pi/180)", AU_AUTO);

colB.ExecuteFormula();



// スクリプト実行前で宣言した変数を使用

Column colC;

colC.Attach(wks, 2);

string strExpression = "cos(Amp*x*pi/180)";
```

```
string strBeforeScript = "double Amp=4.5;" + "¥r¥n" + "range x=col(A);";

string strFormula = strExpression + STR_COL_FORMULAR_SEPARATOR + strBeforeScript;

colC.SetFormula(strFormula, AU_AUTO);

colC.ExecuteFormula();
```

列をソートする

指定した列をソートするには、まず列データをベクトルに取得し、ベクトルでソートした後で値を戻します。列からデータオブジェクトを取得するためのベクトル参照を使用することで、自動的にベクトルを列に付加し、ベクトルのデータが更新されると列に戻すようにします。

```
Worksheet wks = Project.ActiveLayer();

if(!wks)

{

        return;

}

Column col1 = wks.Columns(0);  // 1 列目

vectorbase &v1 = col1.GetDataObject();  // 参照を使用してデータオブジェクトを取得

v1.Sort(SORT_DESCENDING);  // 降順にソート
```

列を逆順にする

列データを逆順にするには、まず列データをベクトルに取得し、ベクトルのデータを逆順にしてから戻します。

```
// 1 列目のデータを逆順にする

Worksheet wks = Project.ActiveLayer();

if(!wks)

{

        return;

}

Column col1 = wks.Columns(0);  // 1 列目

vectorbase &v1 = col1.GetDataObject();  // データオブジェクトを取得

vector<uint> vnIndices;  // 逆順インデックスのためのベクトル

vnIndices.Data(v1.GetSize() - 1, 0, -1);  // 逆順インデックス

v1.Reorder(vnIndices);  // データを逆に
```

列からデータを取得 / 設定

列から数値データ値を取得/設定

```
// 最初の列に付加。列のフォーマっとが

// 数値とテキスト（デフォルト）または数値であるか確認

Column col(wks, 0);



// 列のデータ型を double とする

// 他の数値データ型 int, short, complex などもサポート

vector<double>& vec = col.GetDataObject();



// この列の最後に 100 を追加

vec.Add(100);
```

あるいは、Dataset オブジェクトを使用して列の数値データを取得できます。例えば、

```
Worksheet wks = Project.ActiveLayer();

Dataset ds(wks, 1);

for(int ii=0; ii<ds.GetSize(); ii++)

    out_double("", ds[ii]);
```

列から文字列値を取得/設定

```
Column col(wks, 0); // 1 列目に追加



// 列から文字列配列を取得

vector<string> vs;

col.GetStringArray(vs);



// 列に文字列配列を戻す

vs.Add("test");

col.PutStringArray(vs);
```

列から日時データを取得/設定

列のデータ型が日付や時間の場合、この列から取得したデータは、ユリウス日/時間のデータで、表示-日付-時間-フォーマット文字列ではありません。

```
// アクティブワークシートを取得

Worksheet wks = Project.ActiveLayer();

Column col1(wks, 0);  // 最初の列

Column col2(wks, 1);  // 2 列目

// 列フォーマットが日付や時間であるか確認

if(col1.GetFormat() == OKCOLTYPE_DATE || col1.GetFormat() == OKCOLTYPE_TIME)

{

        // 1 列目からデータを取得, v1 は ユリウス通日データを保持

        vector &v1 = col1.GetDataObject();

        vector &v2 = col2.GetDataObject();  // 2 列目からデータを取得

        v2 = v1;  // 1 列目のユリウス通日データを 2 列目に

        col2.SetFormat(OKCOLTYPE_DATE);  // 2 列目を日付データに

        // 表示フォーマットを MM/dd/yyyy HH:mm:ss にセット

        col2.SetSubFormat(LDF_SHORT_AND_HHMMSS_SEPARCOLON);

}
```

7.3 ワークシート

7.3.1 ワークシート

Origin C には、WorksheetPage 内のワークシートを操作する Worksheet クラスがあります。ワークブックはワークシートのコレクションを含み、ワークシートは Column のコレクションを含みます。Worksheet クラスは、Layer クラスから派生されたものです。

7.3.2 ワークシートの基本操作

ワークシートの基本的な操作には、ワークブックにシートを追加、ワークシートをアクティブ化、ワークシートプロパティの取得や設定、ワークシート削除などをふくみます。ここではいくつかのサンプルも紹介しています。

新しいワークシートを追加する

AddLayer メソッドを使用してワークシートをブックに追加します。

```
// "Book1"というワークブックにアクセス

WorksheetPage wksPage("Book1");
```

```
// ワークブックにシートを追加

int index = wksPage.AddLayer("New Sheet");

// 新しいワークシートにアクセス

Worksheet wksNew = wksPage.Layers(index);
```

ワークシートをアクティブにする

Workbook は、ワークシートを含む Origin のオジェクトです。ワークブック内のシートをアクティブにするには、関数 set_active_layer を使用します。

```
// フルネームによりワークシートにアクセス

Worksheet wks("[Book1]Sheet1");



// このワークシートをアクティブにする

set_active_layer(wks);
```

ワークシートを削除する

Destroy メソッドを使用してワークシートを削除します。

```
Worksheet wks = Project.ActiveLayer();

if( wks ) // アクティブレイヤがワークシートの時

    wks.Destroy(); // ワークシート削除
```

ワークブック内のワークシートにアクセスする

名前でワークシートにアクセスするには、2 通りあります。コンストラクタや付加メソッドにレイヤのフルネームを渡します。レイヤのフルネームには、[ ] 内にページ名が含まれ、そのあとにレイヤ名が続きます。

```
// wksPage は 有効なワークシーページでアクセスしたいシートを保持しているとする

string strFullName = okutil_make_book_sheet_string(wksPage.GetName(), "Sheet1");



// ブックとシートの名前がわかれば、手動で文字列を構築

string strFullName = okutil_make_book_sheet_string("Book5", "Sheet1");
```

全レイヤ名がある場合、ワークシートにアクセスできます。

```
// ワークシートインスタンスを構築して名前の付いたシートを追加

Worksheet wks1(strFullName);
```

```
// 名前の付いたシートに既存のワークシートインスタンスを付加

wks2.Attach(strFullName);
```

ワークシートのコレクションを持つワークブックの場合。**foreach** ステートメントを使用して指定したワークブック内の全ワークシートでループできます。

```
WorksheetPage wksPage("Book1");

foreach(Layer wks in wksPage.Layers)

    out_str(wks.GetName());
```

名前かインデックスで、特定のワークシートにアクセス可能です。

```
//Book1 ページには最低 2 つのシートがあるとする

//これらの名前は Sheet1 と Sheet2

WorksheetPage wksPage("Book1");

Worksheet wksFirst = wksPage.Layers(0); //インデックスによる

Worksheet wksSecond = wksPage.Layers("Sheet2"); //名前による
```

ワークシートの並べ替え

Reorder メソッドでワークブック内のワークシートの順序を変更できます。

```
// このサンプルは、アクティブワークブックに 2 つのシートがあることが前提

// アクティブレイヤからアクティブページを取得

WorksheetPage wksPage;

Worksheet wks = Project.ActiveLayer();

if( wks )

    wksPage = wks.GetPage();



// 2 番目のシートを 1 番目にする

if( wksPage.Reorder(1, 0) )

    out_str("Reorder sheets successfully");
```

ワークシートをコピー

AddLayer メソッドは、1 つのページから他にコピーするのに使用されます。これは、GraphPage, WorksheetPage あるいは MatrixPage をともに使用できます。

次のサンプルは、アクティブフォルダにある全ワークシートをアクティブワークブックにドラッグして統合する方法を示しています。

```
WorksheetPage wksPageDest = Project.Pages();

if( !wksPageDest ) // アクティブウィンドウがない、あるいはアクティブウィンドウはワークシートではない

        return;



bool bKeepSourceLayer = false; // コピー後ソースレイヤは削除

Folder fld = Project.ActiveFolder();

foreach(PageBase pb in fld.Pages)

{

    WorksheetPage wbSource(pb);

    if(!wbSource)

        continue;//ワークブックではない



    if(wbSource.GetName() == wksPageDest.GetName())

        continue;//目的のブックはスキップ



    // 目的のブックにワークシートをコピーし、ソースブックから削除

    foreach(Layer lay in wbSource.Layers)

    {

        Worksheet wks = lay;

        wksPageDest.AddLayer(wks, 0, bKeepSourceLayer);

    }

    wbSource.Destroy();// 空のワークブックを破棄

}
```

ワークシートのフォーマット

プログラミングで、テーマツリーを使用してワークシートをフォーマットできます。以下のサンプルでは、既存テーマツリーを生成して保存する操作を示します。

```
// ワークシートからフォーマットツリーを取得

Worksheet wks = Project.ActiveLayer();
```

```
Tree tr;

tr = wks.GetFormat(FPB_ALL, FOB_ALL, TRUE, TRUE);

out_tree(tr); // スクリプトウィンドウにツリーを出力
```

あるいは、次の 3 つの操作でテーマツリーを行使することもできます。まず、ワークシートを作成し、データを挿入します。

```
// ワークシートを作成

Worksheet wks;

wks.Create("Origin");

wks.SetCell(0, 0, "abc"); // (0, 0) セルにテキストを入力



// フォーマット適用のためにデータ範囲を確立する:

DataRange dr;

int r1 = 0, c1 = 0, r2 = 4, c2 = 1;

dr.Add("Range1", wks, r1, c1, r2, c2);
```

次に、範囲情報を使用してツリーを構成し、希望のプロパティに値を与えます。

```
Tree tr;

// フォーマットを適用したい範囲をセットアップ

tr.Root.RangeStyles.RangeStyle1.Left.nVal = c1 + 1;

tr.Root.RangeStyles.RangeStyle1.Top.nVal = r1 + 1;

tr.Root.RangeStyles.RangeStyle1.Right.nVal = c2 + 1;

tr.Root.RangeStyles.RangeStyle1.Bottom.nVal = r2 + 1;



// 色塗り

tr.Root.RangeStyles.RangeStyle1.Style.Fill.FillColor.nVal = SYSCOLOR_LTCYAN;



// セル内のテキストをアレンジ。2 は中央

tr.Root.RangeStyles.RangeStyle1.Style.Alignment.Horizontal.nVal = 2;



// テキストのフォントサイズ
```

```
tr.Root.RangeStyles.RangeStyle1.Style.Font.Size.nVal = 11;



// テキストの色

tr.Root.RangeStyles.RangeStyle1.Style.Color.nVal = SYSCOLOR_BLUE;
```

そして、データ範囲にフォーマットを適用します。

```
// 指定したデータ範囲にフォーマットを適用

if( 0 == dr.UpdateThemeIDs(tr.Root) ) // 0 はエラーなし

{

        bool bRet = dr.ApplyFormat(tr, true, true);

}
```

セルの統合

Origin C コードを使用して、指定した範囲のワークシートセルを統合できます。選択範囲はデータ領域あるいは、列ラベル領域を指定することができます。ラベルセルを統合したい場合、次のコードで *bLabels* を **true** にします。

```
Worksheet wks;

wks.Create("Origin");



// Grid を定義してワークシートに付加

Grid gg;

gg.Attach(wks);



// 2 列の最初の 2 行を統合する

ORANGE rng;

rng.r1 = 0;

rng.c1 = 0;

rng.r2 = 1;

rng.c2 = 1;



bool bLabels = false;
```

```
bool bRet = gg.MergeCells(rng, bLabels);



if( bRet )

        printf("Successfully merged cells in %s!¥n", wks.GetName());

else

        printf("Failed to merge cells in %s!¥n", wks.GetName());
```

読み取り専用セル

ワークシートセルの内容を変更したくないとき、テーマツリーを使用してセルを読み取り専用にすることができます。

次のサンプルでは、ワークシート内のデータセルを読み取り専用にし、列 1 の 2 つ目のデータセルを編集可能にする方法を示します。

```
// デフォルトテンプレート (Origin) を使用してワークシートを作成

// ロングネーム、単位、コメント行が表示される

Worksheet wks;

wks.Create("Origin");



Tree tr;

tr = wks.GetFormat(FPB_ALL, FOB_ALL, true, true);  // ワークシートのテーマツリー取得



// テーマツリーから指定したツリーノードを取得し、

// データセルを読み取り専用にセットするために開始

string strName = "ogData";  // 希望のフォーマットでノードを取得するために使用

TreeNode trGrid, trNameStyles;

trGrid = tr.Root.Grid;  // グリッドノードを取得

if(!trGrid.IsValid())

        return;



// このノードの子ノードに読み取り専用フォーマットがある

trNameStyles = trGrid.NameStyles;

if(!trNameStyles.IsValid())
```

```
        return;


TreeNode trNameStyle;
bool bRet = false;
// 希望のツリーノードを見つけるためにすべての子ノードをループ
foreach(trNameStyle in trNameStyles.Children)
{
        // "ogData" ノードを検索
        if(0 == trNameStyle.Name.strVal.Compare(strName))
        {
                bRet = true;
                break;
        }
}
if(!bRet)
        return;


trNameStyle.Style.ReadOnly.nVal = 1;  // 読み取り専用にするためにすべてのデータセルをセット


// テーマツリーから特定のツリーノードを取得/作成するため
// 指定したデータセルのお読み取り専用フォーマットをキャンセルするために開始
TreeNode trRangeStyles;
trRangeStyles = trGrid.RangeStyles;  // Grid ノードから RangeStyles ノードを取得
TreeNode trRangeStyle;
if(!trRangeStyles.IsValid())  // RangeStyles ノードがない場合
{
        // RangeStyles ノードを作成
        trRangeStyles = trGrid.AddNode("RangeStyles");
```

```
        // そして RangeStyle1 というサブノードを作成
        trRangeStyle = trRangeStyles.AddNode("RangeStyle1");
}
else  // RangeStyles ノードがある場合
{
        // 子ノードがいくつあるか検索
        int tagNum = trRangeStyles.Children.Count();
        // RangeStyle# (# = tagNum+1) という名前のサブノードを作成
        trRangeStyle = trRangeStyles.AddNode("RangeStyle"+(tagNum+1));
}
// 設定のための範囲を定義。ここでは範囲 y は 1 つのセル
// 1 から始まる範囲内の左のセル
trRangeStyle.Left.nVal = 1;
// 2 から始まる範囲の上のセル (ラベル行を含む)
// 4 つのラベル行があるので、5 は最初のデータセル
trRangeStyle.Top.nVal = 5;
// 1 つのセルなので範囲の右は左と同じ
trRangeStyle.Right.nVal = 1;
// 1 つのセルなので下のセルと上のセルは同じ
trRangeStyle.Bottom.nVal = 5;
trRangeStyle.Style.ReadOnly.nVal = 0;  //0 にして読み取り専用をキャンセル

// ワークシートにフォーマット設定を適用
if(0 == wks.UpdateThemeIDs(tr.Root))
{
        bool bb = wks.ApplyFormat(tr, true, true);
        if(bb)
        {
```

```
                printf("Cell 1 in column 1 is editable.¥n");

        }

}
```

ラベル行のセルを読み取り専用にすることもできます。上述のコードに簡単な変更を加えることで行えます。たとえば、列 2 以外のコメント行を読み取り専用にする場合、以下のようにして変更を行います。

```
/* 上述コード内の以下のラインをコメントアウト

string strName = "ogData";

*/

// これはデータのための行なので、次のようにコメントのための行に変更

string strName = "ogComment";



/* 上記コードの次の 4 行をコメントアウト

trRangeStyle.Left.nVal = 1;

trRangeStyle.Top.nVal = 5;

trRangeStyle.Right.nVal = 1;

trRangeStyle.Bottom.nVal = 5;

*/

// この 4 行は 2 番目のデータセル（ワークシート内で 3 つのラベル行があると仮定）

// の設定のために使用

// ここではコメントセルに対して設定する

// コメント行は列 2 の 3 番目であると仮定して

// 列 1 ではない

trRangeStyle.Left.nVal =

trRangeStyle.Right.nVal = 2;  // 列 2

// コメント行(ワークシート内で表示された 3 番目の行)

trRangeStyle.Top.nVal =

trRangeStyle.Bottom.nVal = 3;
```

7.3.3 ワークシートデータの操作

このセクションでは、Origin C でデータを操作する方法を示すサンプルを紹介します。

ワークシート選択の取得

Worksheet::GetSelectedRange は、ワークシートから 1 つまたは複数の選択したデータを取得するのに使うことができます。次のコードは、ワークシート選択で 1 つの列からデータを取得する方法を示しています。この関数は、1 つの列、1 つの行、ワークシート全体など範囲データ型を返します。

```
Worksheet wks = Project.ActiveLayer();



int r1, c1, r2, c2;

int nRet = wks.GetSelectedRange(r1, c1, r2, c2);



if( WKS_SEL_ONE_COL & nRet ) // 1 つの列だけを選択

{

        // 選択でデータ範囲オブジェクトを構築

        DataRange dr;

        dr.Add("X", wks, r1, c1, r2, c2);



        // 選択した列からデータを取得

        vector vData;

        dr.GetData(&vData, 0);

}
```

ワークシート内の表示範囲を設定する

Worksheet 内の表示範囲を設定する場合、Worksheet::SetBounds を使い、開始行／終了行に設定と同じです。

次のコードは、現在のワークシートウィンドウにすべての列の開始と終了をセットする方法を示します。

```
Worksheet wks = Project.ActiveLayer();



// 行の開始と終了

int begin = 9, end = 19;
```

```
// すべての列の開始と終了をセット

int c1 = 0, c2 = -1; // -1 は終了



wks.SetBounds(begin, c1, end, c2);
```

ワークシートに大きなデータセットを配置する

ワークシート内で大きなデータセット(例:1000 列)を操作するとき、Origin C 関数を効率よく実行するためには、以下のステップを使います。

- ワークシートにデータを配置する列と行を準備します。
- サイズをセットするには、Worksheet::SetSize を使い、Worksheet::AddCol は使いません。
- Origin が新しい列を追加するときに重複した名前を持たないように既存の列のショートネームをチェックする必要があるので、事前に空のワークシートでサイズをセットし、列と行の属性は無しにします。これにより時間が削減できます。

  while(wks.DeleteCol(0)); を使って、すべての列を削除して、空のワークシートを作成します。
- DataObject::GetInternalDataBuffer を使って、バッファでワークシート列にデータを配置します。
- 実行スピードを上げるために関数を実行しているとき、コードビルダは閉じておきます。

以下のサンプルコードをご覧ください。

```
// ワークシートの準備

Worksheet wks;

wks.Create("Origin");

while( wks.DeleteCol(0) );

int rows = 100, cols = 1000;

wks.SetSize(rows, cols);



// ワークシート列に 1 つずつデータを配置

foreach(Column col in wks.Columns)

{

    col.SetFormat(OKCOLTYPE_NUMERIC);

    col.SetInternalData(FSI_SHORT);

    col.SetUpperBound(rows-1);// 最後の行のインデックス, 0 オフセット
```

```
    int     nElementSize;
    uint    nNum;
    LPVOID  pData = col.GetInternalDataBuffer(&nElementSize, &nNum);
    short* psBuff = (short*)pData;

    // OC のループは遅いですが DLL にポインタを渡すと
    // 操作が高速になる。ここではポインタの動作の表示のみ
    for(int ii = 0; ii < rows; ii++, psBuff++)
    {
        *psBuff = (ii+1) * (col.GetIndex()+1);
    }
    col.ReleaseBuffer(); // この呼び出しを記録しない
}
```

ワークシートの埋め込みグラフにアクセスする

新しいグラフと新しいワークシートを作成し、1 つのワークシートセルにグラフを埋め込みます。

```
GraphPage gp;
gp.Create("Origin");

Worksheet wks;
wks.Create();

int nOptions = EMBEDGRAPH_KEEP_ASPECT_RATIO | EMBEDGRAPH_HIDE_LEGENDS;

// ワークシートセル(0, 0)にグラフを配置
wks.EmbedGraph(0, 0, gp, nOptions);
```

ワークシート内に埋め込んだグラフに名前またはインデックスでアクセスします。

```

```

```
// アクティブワークシートから埋め込みグラフを取得

Worksheet wks = Project.ActiveLayer();



GraphPage gp;

gp = wks.EmbeddedPages(0); //  埋め込みグラフページをインデックスで取得



gp = wks.EmbeddedPages("Graph1"); // 埋め込みグラフページを名前で取得
```

ワークシートデータのソート

Sort メソッドを使って列データをソートすることができます。1 つの列をソートし、vectorbase::Sort メソッドを使います。

```
// 列のソート

// 実行前に 2 列を持つアクティブワークシートにデータを入力

// 例えば、¥Samples¥Mathematics¥Sine Curve.dat をワークシートにインポート

Worksheet wks = Project.ActiveLayer();

Column colY(wks, 1); // Y column



// ソートした後、 (x, y)の元の関係は崩れる

vectorbase& vec = colY.GetDataObject();

vec.Sort();
```

ワークシート内のすべての列をソートするには、Worksheet::Sort メソッドを使います。

```
// ワークシートのソート

// 実行前に 2 列を持つアクティブワークシートにデータを入力

// 例えば、¥Samples¥Mathematics¥Sine Curve.dat をワークシートにインポート

Worksheet wks = Project.ActiveLayer();



int nCol = 1; //  2 番目の列にすべてのワークシートデータを昇順ソート

BOOL bIsAscending = true;

BOOL bMissingValuesSmall = TRUE; // 欠損値を最小値として取り扱う

int r1 = 0, c1 = 0, r2 = -1, c2 = -1; //  -1 は r2 と c2 の最後
```

```
// ソート後、各(x, y)は元の関係を維持

wks.Sort(nCol, bIsAscending, bMissingValuesSmall, r1, c1, r2, c2);
```

ワークシートデータをマスクする

次のコードは、指定した列に対して、0と等しいか、それより小さいデータ行にマスクをセットする方法を示します。

```
int nCol = 1;

Worksheet wks = Project.ActiveLayer();

Column col(wks, nCol);

vector vData = col.GetDataObject();



// 0と等しいか、それより小さいすべてのデータを見つけて、インデックスを返す

vector<uint> vnRowIndex;

vData.Find(MATREPL_TEST_LESSTHAN | MATREPL_TEST_EQUAL, 0, vnRowIndex);



// 行と列のインデックスで追加される複数の部分範囲を含む範囲を構築

DataRange dr;

for(int nn = 0; nn < vnRowIndex.GetSize(); nn++)

{

        int r1, c1, r2, c2;

        r1 = r2 = vnRowIndex[nn];

        c1 = c2 = nCol;

        dr.Add("X", wks, r1, c1, r2, c2);

}



// データ範囲にマスクをセット

dr.SetMask();
```

サイズをセット

Worksheet::SetSize メソッドを使用して、ワークシートの行と列の数をセットできます。

```
// 行と列の数をセット。データは保持

// 一度に多くの列が行を追加したい場合 SetSize を使用した方が良い

int nNumRows = 100;

int nNumCols = 20;

wks.SetSize(nNumRows, nNumCols);

// 行の数を変更して列の数は変更しない場合

// 代わりに -1 を使用例えば、

wks.SetSize(nNumRows, -1);

// 列の数は変更して行は変更しない場合も同様
```

ワークシートデータを削減する

ワークシート内の XY データの削減のために、Origin C いくつかの関数を提供してます。例えば、X 増分による XY データの削減には ocmath_reducexy_fixing_increbin、グループの数による XY データの削減には ocmath_reducexy_n_groups、N ポイントごとに XY データを削減するには ocmath_reducexy_n_points を使用します。ここでは、N ポイントごとに XY データを削減する方法を、以下のサンプルで示します。

```
Worksheet wks = Project.ActiveLayer();  // アクティブワークシートを取得

if(!wks)

{

        return;

}

Column colX(wks, 0);  // ワークシートの 1 列目

Column colY(wks, 1);  // ワークシートの 2 列目

if(colX && colY)

{

        vectorbase &vbInterY = colY.GetDataObject();  // Y 列データを取得

        vector vY = vbInterY;

        vector vReduced(vY.GetSize());

        int nPoints = 3;

        // 3 ポイントごとに削減。結果は各 3 ポイントの平均とする
```

```
        int nNewSize = ocmath_reducexy_n_points(vY, vReduced, vY.GetSize(),
                nPoints, REDUCE_XY_STATS_MEAN);
        int iReduced = wks.AddCol("Reduced");  // 結果のために新しい列を追加
        Column colReduced(wks, iReduced);
        vectorbase &vbReduced = colReduced.GetDataObject();
        vbReduced = vReduced;
}
```

LT 条件でワークシートからデータを抽出

Worksheet::SelectRows メソッドを使ってワークシートデータを選択します。行は、複数の列をまたがって選択できます。

```
// 条件を元にしたワークシートからデータを選択
// 'uint'型の vector に選択した行のインデックスを配置
Worksheet wks = Project.ActiveLayer();

// 条件式に基づくワークシートデータをチェックし
// 行インデックスを'vnRowIndices'に出力
// LabTalk の範囲オブジェクト 'a' = column 1, 'b' = column 2 を定義
string strLTRunBeforeloop = "range a=1; range b=2";
string strCondition = "abs(a) >= 1 && abs(b) >= 1";
vector<uint> vnRowIndices; // 出力
int r1 = 0, r2 = -1; // 行範囲, -1 は r2 の最後の行

// 任意の最大行数, -1 は制限なし

int nMax = -1;

int num = wks.SelectRows(strCondition, vnRowIndices, r1, r2, nMax,
                strLTRunBeforeloop);
```

選択をハイライトする 2 つの方法があります。最初の方法は選択したインデックスをハイライトする方法です。

```
//  方法 1 : vnRowIndices で行をハイライト

Grid gg;

if( gg.Attach(wks) )

{

        // uint 型 の vector データを int 型の vector に変換

    vector<int> vnRows;

    vnRows = vnRowIndices;



    gg.SetSelection(vnRows);

}
```

データ選択をハイライトする 2 つ目の方法は、選択した行の塗り色を指定する方法です。

```
// 方法 2 :  vnRowIndices で選択した行の色を塗りつぶす

DataRange dr;



// vnRowIndices で行インデックスでデータ範囲を構築

for(int index=0; index<vnRowIndices.GetSize(); index++)

{

        // すべての列に対して 0(最初の列) および -1(最後の列)

        // 範囲名の変数に対して""であればデフォルト名を使用

        dr.Add("", wks, vnRowIndices[index], 0, vnRowIndices[index], -1);

}



Tree tr;

tr.Root.CommonStyle.Fill.FillColor.nVal = SYSCOLOR_BLUE; // 塗りつぶし色 = 青

tr.Root.CommonStyle.Color.nVal = SYSCOLOR_WHITE; // フォントの色 = 白



if( 0 == dr.UpdateThemeIDs(tr.Root) ) // エラーなしで 0 を返す

{
```

```
        bool bRet = dr.ApplyFormat(tr, true, true);

}
```

2 つのワークシート内のデータを比較する

2 つのワークシートの行数または列数を比較したり、データ自体を比較するのに役立ちます。Datasheet::GetNumRows および Datasheet::GetNumCols メソッドを使って、ワークシートから行や列をカウントします。

```
if( wks1.GetNumRows() != wks2.GetNumRows()

        || wks1.GetNumCols() != wks2.GetNumCols() )

{

        out_str("The two worksheets are not the same size");

        return;

}
```

似たような操作を実行する別の方法は、それぞれのワークシートのデータを vector にコピーし、それぞれの vector データを比較します。

```
// ワークシート 1 の列から 1 つずつすべてのデータを取得

vector vec1;

foreach(Column col in wks1.Columns)

{

        vector& vecCol = col.GetDataObject();

        vec1.Append(vecCol);

}



// ワークシート 2 の列から 1 つずつすべてのデータを取得

vector vec2;

foreach(col in wks2.Columns)

{

        vector& vecCol = col.GetDataObject();

        vec2.Append(vecCol);

}
```

```
if( vec1.GetSize() != vec2.GetSize() )
{
        out_str("The size of the two data sets is not equal");
        return;
}
```

データ要素自体を比較するには、上記サンプルで vector に対して、ocmath_compare_data 関数を使います。

```
bool bIsSame = false;
double dTolerance = 1e-10;
ocmath_compare_data(vec1.GetSize(), vec1, vec2, &bIsSame, dTolerance);
if( bIsSame )
{
        out_str("Data in the two worksheets are the same");
}
```

7.3.4 ワークシートから行列に変換

分析またはグラフ作成時に、ワークシートから行列に変換またはその反対の操作をして、データを再構成する必要があるかもしれません。このページはワークシートを行列に変換する時のサンプルと情報を紹介します。なお、反対の操作をする際は行列をワークシートに変換するを確認してください。

ワークシートグリッディング

1. コマンドウィンドウで次のコマンドを実行し、nag_utils.c ファイルをコンパイルし、現在のワークスペースにそれを追加します。

```
Run.LoadOC(Originlab¥nag_utils.c, 16);
```

2. Origin C ファイルにヘッダファイルを含めます。

```
#include <wks2mat.h>

#include <Nag_utils.h>
```

3. アクティブなワークシート XYZ 列から XYZ データを取得します。

```
// XYZ 列から XYZ データ範囲を取得
```

```
XYZRange rng;
rng.Add(wks, 0, "X");
rng.Add(wks, 1, "Y");
rng.Add(wks, 2, "Z");

// データ範囲オブジェクトから XYZ データをベクターに取得
vector vX, vY, vZ;
rng.GetData(vZ, vY, vX);
```

4. 元のデータ型を調査します。例:XY 等間隔配置、疎データ

```
UINT nVar;
double xmin, xstep, xmax, ymin, ystep, ymax;
int nSize = vX.GetSize();
int nMethod = ocmath_xyz_examine_data(nSize, vX, vY, vZ, 1.0e-8, 1.0e-8,
&nVar, &xmin, &xstep, &xmax, &ymin, &ystep, &ymax);
```

5. 結果の行列ウィンドウに対する行と列の数を計算します。

```
int nRows = 10, nCols = 10;
if( 0 == nMethod || 1 == nMethod ) // XY 等間隔または疎データ
{
        double dGap = 1.5;
        if( !is_equal(ystep, 0) )
                nRows = abs(ymax - ymin)/ystep + dGap;
```

```
        if( !is_equal(xstep, 0) )

                nCols = abs(xmax - xmin)/xstep + dGap;

}
```

6. 結果の行列ウィンドウを準備します。

```
// グリッディング結果を配置する行列ウィンドウを準備

MatrixPage mp;

mp.Create("origin"); // 行列ウィンドウを作成

MatrixLayer ml = mp.Layers(0); // 第 1 行列シートを取得

MatrixObject mo(ml, 0); // 第 1 行列オブジェクトを取得



mo.SetXY(xmin, ymin, xmax, ymax); // X と Y の開始/終了値をセット

mo.SetSize(nRows, nCols); // 行と列の数をセット
```

7. 異なるメソッドで XYZ グリッディングを行います。

```
matrix& mat = mo.GetDataObject(); // 行列オブジェクトからデータオブジェクトを取得



int iRet;

switch(nMethod)

{

case 0:// XY 等間隔

        iRet = ocmath_convert_regular_xyz_to_matrix(nSize, vX, vY, vZ,

                mat, xmin, xstep, nCols, ymin, ystep, nRows);

        printf("--- %d: regular conversion ---¥n", iRet);

        break;
```

```
case 1:// 疎データ
        iRet = ocmath_convert_sparse_xyz_to_matrix(nSize, vX, vY, vZ,
                mat, xmin, xstep, nCols, ymin, ystep, nRows);
        printf("--- %d: sparse conversion ---¥n", iRet);
        break;

case 2:// ランダム(Renka Cline)
        vector vxGrid(nRows*nCols), vyGrid(nRows*nCols);
        iRet = ocmath_mat_to_regular_xyz(NULL, nRows, nCols, xmin,
                xmax, ymin, ymax, vxGrid, vyGrid);
        if( iRet >= 0 )
        {
                iRet = xyz_gridding_nag(vX, vY, vZ, vxGrid, vyGrid, mat);
        }
        printf("--- %d: random conversion ---¥n", iRet);
        break;

default:// エラー
        printf("--- Error:Other method type ---¥n");
}
```

ワークシートから行列

ワークシートに含まれるデータは、一連の関数を使って行列に変換することができます。

行列のようなワークシートデータを直接行列に変換するために、ワークシートの最初の列と行に X または Y の値を入力します。しかし、行列の座標は等間隔でなければならいので、元のワークシートに等間隔な X/Y 値が入力されている必要があります。メソッド CopyFromWks を直接使用するか、単に行列に XYZ データ範囲を付けます。

次のサンプルはワークシートから直接変換する方法を示しています。

```
// 方法 1: CopyFromWks を使用
Worksheet wks = Project.ActiveLayer();
if(!wks)
```

```
{
        return;
}
MatrixPage matPg;
matPg.Create("Origin");
MatrixLayer matLy = matPg.Layers(0);
Matrix mat(matLy);


matrix<double> mat1;
if(!mat1.CopyFromWks(wks, 1, -1, 1, -1))
{
        out_str("Error:CopyFromWks failed!");
        return;
}
mat = mat1;



// 方法 2: 行列オブジェクトに付加
Worksheet wks = Project.ActiveLayer();
if(!wks)
{
        return;
}
int nCols = wks.GetNumCols();
int nRows = wks.GetNumRows();
DataRange dr;
dr.Add("X", wks, 0, 1, 0, nCols - 1);  // 最初のセルを除いた最初の行
dr.Add("Y", wks, 1, 0, nRows - 1, 0);  // 最初のセルを除いた最初の列
```

```
dr.Add("Z", wks, 1, 1, nRows - 1, nCols - 1);

MatrixPage matPg;

matPg.Create("Origin");

MatrixLayer matLy = matPg.Layers(0);

MatrixObject mo = matLy.MatrixObjects(0);

MatrixObject moTmp;

moTmp.Attach(dr);

matrixbase &matTmp = moTmp.GetDataObject();

matrixbase &mat = mo.GetDataObject();

mat = matTmp;

moTmp.Detach();
```

ワークシートデータが XYZ 列形式で構成されているとき、グリッディング法を使ってこのようなデータを行列に変換します。多くのグリッディング法が利用でき、これにより元のデータを補間し、指定した次数での等間隔な XY で、値の配列を生成します。

次のサンプルでは、Renka-Cline の手法により XYZ データを変換します。

```
// Renka-Cline グリッディング手法によりワークシートデータを 20*20 の行列に変換

Worksheet wks = Project.ActiveLayer();

if(!wks)

{

        return;

}

Dataset dsX(wks, 0);

Dataset dsY(wks, 1);

Dataset dsZ(wks, 2);

int nPoints = dsX.GetSize();

vector vX = dsX;

vector vY = dsY;

vector vZ = dsZ;
```

```
ocmath_RenkaCline_Struct comm;

ocmath_renka_cline_interpolation(nPoints, vX, vY, vZ, &comm);



// グリッディングの X と Y をセット

double dXMin, dXMax, dYMin, dYMax;

vX.GetMinMax(dXMin, dXMax);

vY.GetMinMax(dYMin, dYMax);


// Kriging アルゴリズムを使用したランダム行列変換

int nRows = 20;

int nCols = 20;

matrix mZ(nRows, nCols);

vector vEvalX(nRows * nCols);

vector vEvalY(nRows * nCols);

ocmath_mat_to_regular_xyz(NULL, nRows, nCols, dXMin, dXMax, dYMin, dYMax, vEvalX, vEvalY, NULL, true);

ocmath_renka_cline_eval(&comm, nRows * nCols, vEvalX, vEvalY, mZ);

ocmath_renka_cline_struct_free(&comm);


// 行列を作成して結果を格納

MatrixLayer    mResultLayer;

mResultLayer.Create();

Matrix    matResult(mResultLayer);

matResult = mZ;

MatrixObject mo = mResultLayer.MatrixObjects(0);

mo.SetXY(dXMin, dYMin, dXMax, dYMax);// 行列の X と Y 範囲を設定
```

7.3.5 仮想行列

ワークシートウィンドウから仮想行列を構築することができます。仮想行列の XYZ データに対するワークシートのデータ範囲を別々に取得します。X および Y データを指定しない場合、デフォルトデータが自動的に使われます。次のコードはアクティブワークシートウィンドウから仮想行列を構築し、この仮想行列をグラフにプロットする方法を示しています。

```
// 実行前に、アクティブワークシートにデータがあることを確認

// 例えば、新しいワークシートに、Origin のインストールフォルダの

// Samples¥Matrix Conversion and Gridding にある XYZ Random Gaussian.dat をインポート

Worksheet wks = Project.ActiveLayer();



int r1, r2;

int c1 = 0, c2 = 2;

wks.GetBounds(r1, c1, r2, c2);



// Z データのみデータ範囲を構築すると

// X、Y データは自動的に割り当て

DataRange dr;

dr.Add("Z", wks, r1, c1, r2, c2);



MatrixObject mo;

mo.Attach(dr);



int nRows = mo.GetNumRows();

int nCols = mo.GetNumCols();



// デフォルトの x, y 範囲を取得

double xmin, xmax, ymin, ymax;

mo.GetXY(xmin, ymin, xmax, ymax);



GraphPage gp;
```

```
gp.Create("CONTOUR");

GraphLayer gl = gp.Layers(0);



gl.AddPlot(mo, IDM_PLOT_CONTOUR);

gl.Rescale();



mo.Detach();
```

XY データを割り当てたい場合、データは単調増加あるいは、減少するものである必要があります。次のサンプルは、XYZ データ範囲から仮想行列を構築する方法を示しています。

```
// アクティブレイヤには 5 列のデータがあるとする

Worksheet wks = Project.ActiveLayer();



// 列 0 から 4 に対する最小と最大の行インデックスを取得

int r1, r2, c1 = 0, c2 = 4;

wks.GetBounds(r1, c1, r2, c2);



// XYZ データを持つデータ範囲オブジェクトを作成

DataRange dr;

dr.Add("X", wks, 0, 1, 0, c2); // 最初のセルを除く最初の行

dr.Add("Y", wks, 1, 0, r2, 0); // 最初のセルを除く最初の列

dr.Add("Z", wks, 1, 1, r2, c2);



MatrixObject mo;

mo.Attach(dr);
```

8 グラフ

GraphPage クラスはグラフウィンドウに対して操作するものです。各グラフウィンドウに GraphPage オブジェクトがあります。GraphPage オブジェクトは、複数レイヤを含みます。これらのレイヤは、**GraphLayer** オブジェクトです。

既存グラフにアクセスする

既存のグラフにアクセスする方法が複数あります。使用する方法は、ワークブックや行列ブックに対して使われるのと同じです。

グラフの名前をクラスコンストラクタに渡してグラフにアクセスできます。

```
GraphPage grPg("Graph1");

if( grPg ) // "Graph1"というグラフがあれば

    grPg.SetName("MyGraph1"); // 名前を変更
```

Project クラスは、プロジェクト内のすべてのグラフのコレクションを含みます。次の例は、グラフコレクションをループし、各グラフの名前を出力する方法を示しています。

```
foreach(GraphPage grPg in Project.GraphPages)

    out_str(grPg.GetName()); // グラフ名を出力
```

Collection クラスの **Item** メソッドにグラフの 0 から始まるインデックスを渡して、グラフにアクセスできます。

```
GraphPage grPg;

grPg = Project.GraphPages.Item(2);

if( grPg ) //  3 番目のグラフがあれば

    out_str(grPg.GetName()); // グラフ名を出力
```

グラフを削除する

すべての Origin C の内部クラスは、**OriginObject** クラスから派生したものです。このクラスは、オブジェクトを破棄するのに使用する **Destroy** メソッドがあります。グラフに対してこのメソッドを呼ぶとグラフ、グラフ内のすべてのレイヤ、各レイヤのすべてのグラフオブジェクトを削除します。

```
GraphPage grPg;

grPg = Project.GraphPages.Item(0); // プロジェクト内の最初のグラフを取得

if( grPg ) // グラフがあれば

    grPg.Destroy(); // グラフを削除
```

8.1 グラフの作成と編集

8.1.1 グラフウィンドウの作成

Create メソッドを使って新しいグラフを作成します。

```
GraphPage gp;

gp.Create("3D"); // 3D テンプレートを使ってグラフを作成
```

8.1.2 グラフページのフォーマットを取得

```
GraphPage gp("Graph1");



Tree tr;

tr = gp.GetFormat(FPB_ALL, FOB_ALL, true, true);

out_tree(tr);
```

8.1.3 グラフページフォーマットをセット

次のサンプルコードは、2 色のグラデーションでページの背景色をセットする方法を示しています。

```
Tree tr;

tr.Root.Background.BaseColor.nVal = SYSCOLOR_RED;

tr.Root.Background.GradientControl.nVal = 1;

tr.Root.Background.GradientColor.nVal = SYSCOLOR_BLUE;



GraphPage gp("Graph1");

if(0 == gp.UpdateThemeIDs(tr.Root) )

        gp.ApplyFormat(tr, true, true);
```

8.1.4 グラフレイヤのフォーマットを取得

```
GraphLayer gl = Project.ActiveLayer();



Tree tr;

tr = gl.GetFormat(FPB_ALL, FOB_ALL, true, true);

out_tree(tr);
```

8.1.5　グラフレイヤのフォーマットをセット

以下のサンプルコードは、グラフレイヤオブジェクトの背景を黒線フォーマットにセットする方法を示します。

```
GraphLayer gl = Project.ActiveLayer();



Tree tr;

tr.Root.Background.Border.Color.nVal = SYSCOLOR_BLACK;

tr.Root.Background.Border.Width.nVal = 1;

tr.Root.Background.Fill.Color.nVal = SYSCOLOR_WHITE;



if( 0 == gl.UpdateThemeIDs(tr.Root) )

        gl.ApplyFormat(tr, true, true);
```

8.1.6　追加の線を表示

このサンプルは、追加の線、Y=0/X=0 での線、反対の線を表示する方法を示しています。

```
GraphLayer gl = Project.ActiveLayer();

Axis axesX = gl.XAxis;



axesX.Additional.ZeroLine.nVal = 1; // Y = 0 の線を表示

axesX.Additional.OppositeLine.nVal = 1; // X 軸の反対の線を表示
```

8.1.7　グリッド線を表示

このサンプルは、グリッド線の表示方法および色付けの方法を示しています。

色の値は、Origin の内部のカラーパレットまたは RGB 値へのインデックスにすることができます。色の値についての詳細は、データ型と変数 の 色 をご覧ください。

```
GraphLayer gl = Project.ActiveLayer();

Axis axisY = gl.YAxis;

Tree tr;



//  主グリッド表示

TreeNode trProperty = tr.Root.Grids.HorizontalMajorGrids.AddNode("Show");

trProperty.nVal = 1;
```

```
tr.Root.Grids.HorizontalMajorGrids.Color.nVal = RGB2OCOLOR(RGB(100, 100, 220));
tr.Root.Grids.HorizontalMajorGrids.Style.nVal = 1; // 実線
tr.Root.Grids.HorizontalMajorGrids.Width.dVal = 1;

// 副グリッド表示
trProperty = tr.Root.Grids.HorizontalMinorGrids.AddNode("Show");
trProperty.nVal = 1;
tr.Root.Grids.HorizontalMinorGrids.Color.nVal = SYSCOLOR_GREEN; // 緑
tr.Root.Grids.HorizontalMinorGrids.Style.nVal = 2; // ドット
tr.Root.Grids.HorizontalMinorGrids.Width.dVal = 0.3;

if(0 == axisY.UpdateThemeIDs(tr.Root) )
{
        bool bRet = axisY.ApplyFormat(tr, true, true);
}
```

8.1.8　軸スケールの設定

このサンプルは、スケールのパラメータ、増分、スケールタイプなどをセットする方法を示しています。

```
GraphLayer gl = Project.ActiveLayer();
Axis axesX = gl.XAxis;

axesX.Scale.From.dVal = 0;
axesX.Scale.To.dVal = 1;
axesX.Scale.IncrementBy.nVal = 0; // 0=値で増加; 1=主目盛の数
axesX.Scale.Value.dVal = 0.2; // 増分
axesX.Scale.Type.nVal = 0;// 線形
axesX.Scale.Rescale.nVal = 0; // 再スケールタイプ
axesX.Scale.RescaleMargin.dVal = 8; // 8 パーセント
```

次のサンプルは、Y 軸の主目盛数をセットする方法を示しています。

```
GraphLayer gl = Project.ActiveLayer();

Axis axesY = gl.YAxis;



axesY.Scale.IncrementBy.nVal = 1; // 0: 値で増加; 1: 主目盛の数

axesY.Scale.MajorTicksCount.nVal = 5;
```

8.1.9 軸フォーマットを取得

```
GraphLayer gl = Project.ActiveLayer();

Axis axisX = gl.XAxis;



// 軸フォーマット設定をツリーに取得

Tree tr;

tr = axisX.GetFormat(FPB_ALL, FOB_ALL, true, true);

out_tree(tr);
```

8.1.10 軸ラベルを設定

軸ラベルは、通常のテキストオブジェクトであり、GraphObject クラスを使って Origin C でアクセスできます。デフォルトグラフでは、X 軸は XB という名前で、Y 軸は YL という名前になっています。次のコードは、X および Y 軸ラベルにアクセスする方法を示し、デフォルトグラフがアクティブページとします。

```
GraphLayer gl = Project.ActiveLayer();    // アクティブグラフレイヤ取得

GraphObject grXL = gl.GraphObjects("XB"); // X 軸ラベルを取得

GraphObject grYL = gl.GraphObjects("YL"); // Y 軸ラベルを取得
```

これで、軸ラベルにアクセスし、値を変更できます。次のコードは、X 軸ラベルを直接セットし、Y 軸ラベルを LabTalk 文字列変数にリンクすることで間接的にセットします。LabTalk 変数へのリンクは、ラベルのプログラミング制御オプション"変数へリンク"を有効にする必要があります。このオプションはデフォルトで有効になっています。

```
grXL.Text = "My New X Asis Label";

LT_set_str("abc$", "My String Variable");

grYL.Text = "%(abc$)";
```

ラベルが変わったことを確認するには、グラフページをリフレッシュする必要があります。GraphLayer オブジェクトを使って、次のコードでページをリフレッシュできます。

```
gl.GetPage().Refresh();
```

8.1.11 上 X 軸を表示

このサンプルは、上 X 軸を表示する方法を示します。

```
// 軸と刻みを表示

Tree tr;

TreeNode trProperty = tr.Root.Ticks.TopTicks.AddNode("Show");

trProperty.nVal = 1;



// 軸ラベルを表示

trProperty = tr.Root.Labels.TopLabels.AddNode("Show");

trProperty.nVal = 1;



GraphLayer gl = Project.ActiveLayer();

Axis axesX = gl.XAxis;

if(0 == axesX.UpdateThemeIDs(tr.Root) )

{

        bool bRet = axesX.ApplyFormat(tr, true, true);

}
```

8.1.12 軸刻みを編集

このサンプルは、軸ダイアログ->軸の主要設定(タイトル・形態)でフォーマットをセットする方法を示しています。

```
GraphLayer gl = Project.ActiveLayer();

Axis axesX = gl.XAxis;



Tree tr;

// データプロットの色を従って、刻みの色を自動にセット

tr.Root.Ticks.BottomTicks.Color.nVal = INDEX_COLOR_AUTOMATIC;

tr.Root.Ticks.BottomTicks.Width.dVal = 3;

tr.Root.Ticks.BottomTicks.Major.nVal = 0; // 0:刻みを両側表示

tr.Root.Ticks.BottomTicks.Minor.nVal = 2; // 2:外側
```

```
tr.Root.Ticks.BottomTicks.Style.nVal = 0; // 実線



if(0 == axesX.UpdateThemeIDs(tr.Root) )

    bool bRet = axesX.ApplyFormat(tr, true, true);
```

8.1.13 軸目盛ラベルを編集

このサンプルは、指定した位置で軸ラベルをセットする方法を示します。軸ダイアログの軸目盛のカスタム化タブで同じ操作を行うことができます。

```
GraphLayer gl = Project.ActiveLayer();

Axis axesX = gl.XAxis;



Tree tr;

// スケール値として軸の開始と終了を表示

tr.Root.Labels.BottomLabels.Custom.Begin.Type.nVal = 2;

tr.Root.Labels.BottomLabels.Custom.End.Type.nVal = 2;



// 特別な値とテキストを特定のポイントにセット

tr.Root.Labels.BottomLabels.Custom.Special.Type.nVal = 3;

tr.Root.Labels.BottomLabels.Custom.Special.Label.strVal = "Mid";

tr.Root.Labels.BottomLabels.Custom.Special.Value.dVal = 12;



if(0 == axesX.UpdateThemeIDs(tr.Root) )

{

        bool bRet = axesX.ApplyFormat(tr, true, true);

}
```

8.2 データプロットを追加する

プロット または **データプロット** は、グラフレイヤ内のデータ表示です。各グラフレイヤには、1 つ以上のプロットが含まれます。

8.2.1 2D グラフ (XY, Y エラー, 縦棒/横棒)

XY 散布図をプロット

次のコードは、アクティブワークシートから XYYErr データ範囲を構築する方法を示し、新しく作成したグラフでデータ範囲をプロットします。

```
Worksheet wks = Project.ActiveLayer();



// 範囲名は、X, Y, Z, ED(YErr)にする



DataRange dr;

dr.Add(wks, 0, "X"); // 第 1 列は X データ

dr.Add(wks, 1, "Y"); // 第 2 列は Y データ

dr.Add(wks, 2, "ED"); // 第 1 列は Y エラーデータ(任意)



// グラフウィンドウを作成

GraphPage gp;

gp.Create("Origin");

GraphLayer gl = gp.Layers(); // アクティブレイヤを取得



// XY データ範囲を散布図としてプロット

// IDM_PLOT_SCATTER はプロットタイプ ID で、他のタイプの ID は oPlotIDs.h ファイルを確認

int nPlotIndex = gl.AddPlot(dr, IDM_PLOT_SCATTER);

// プロットインデックス(オフセットは 0)を返す、エラーは-1 を返す

if( nPlotIndex >= 0 )

{

        gl.Rescale(); // 軸を再スケールしすべてのデータポイントを表示

}
```

Y エラープロットを付ける

YErr データを既存の XY データプロットに付加します。

```
GraphLayer gl = Project.ActiveLayer();

DataPlot dp = gl.DataPlots(-1);   // アクティブデータプロットを取得



// Y エラー列を取得

WorksheetPage wksPage("Book1");

Worksheet wks = wksPage.Layers();

Column  colErrBar(wks, 2);



// アクティブデータプロットに Y エラー列をプロット

Curve   crv(dp);

int               nErrPlotIndex = gl.AddErrBar(crv, colErrBar);

out_int("nErrPlotIndex = ", nErrPlotIndex);
```

縦棒 / 横棒グラフ

```
// 実行前にアクティブウィンドウがワークシートであることを確認

Worksheet wks = Project.ActiveLayer();

DataRange dr;

dr.Add(wks, 1, "Y"); // 1 列を持つデータ範囲を構築

GraphPage gp;

gp.Create("BAR"); // 指定したテンプレートでグラフを作成

GraphLayer gl = gp.Layers(-1); // アクティブグラフレイヤを取得

int index = gl.AddPlot(dr, IDM_PLOT_BAR);

if( index >= 0 )

{

        out_str("Plot bar");

        gl.Rescale();

}
```

8.2.2 3D グラフ

行列からグラフウィンドウに 3D 曲面図をプロットします。

```
// 行列データを用意

MatrixLayer ml;

string strFile = GetAppPath(true) + "Samples¥¥Matrix Conversion and Gridding¥¥

2D Gaussian.ogm";

ml.Open(strFile);

MatrixObject mo = ml.MatrixObjects(0);



// テンプレートでグラフページを作成

GraphPage gp;

gp.Create("CMAP");

GraphLayer gl = gp.Layers(0);



// 3D 曲面図をプロット

int nPlotIndex = gl.AddPlot(mo, IDM_PLOT_SURFACE_COLORMAP);

if(0 == nPlotIndex)

{

        gl.Rescale();

        printf("3D Surface plotted successfully¥n");

}
```

8.2.3 等高線図

XYZ 等高線図をプロット

```
// 実行前に、アクティブワークシートウィンドウに XYZ データが

// あることを確認。または、¥Samples¥Matrix Conversion and Gridding¥

// XYZ Random Gaussian.dat をワークシートにインポート

Worksheet wks = Project.ActiveLayer();

DataRange dr;
```

```
dr.Add(wks, 0, "X");
dr.Add(wks, 1, "Y");
dr.Add(wks, 2, "Z");

// テンプレートでグラフページを作成
GraphPage gp;
gp.Create("TriContour");
GraphLayer gl = gp.Layers();

// タイプ ID で XYZ 等高線をプロット
int nPlot = gl.AddPlot(dr, IDM_PLOT_TRI_CONTOUR);
if( nPlot >= 0 )
{
        gl.Rescale();
        printf("XYZ contour plotted successfully¥n");
}
```

色付き等高線をプロット

```
MatrixLayer ml = Project.ActiveLayer();
MatrixObject mo = ml.MatrixObjects(0);
// テンプレートでグラフウィンドウを作成
GraphPage gp;
gp.Create("contour");
GraphLayer gl = gp.Layers();
int nPlot = gl.AddPlot(mo, IDM_PLOT_CONTOUR);
if( nPlot >= 0 )
{
        gl.Rescale();
}
```

8.2.4 イメージプロット

```
MatrixLayer ml = Project.ActiveLayer();

MatrixObject mo = ml.MatrixObjects(0);



// テンプレートでグラフウィンドウを作成

GraphPage gp;

gp.Create("image");

GraphLayer gl = gp.Layers();



int nPlot = gl.AddPlot(mo, IDM_PLOT_MATRIX_IMAGE);

if( nPlot >= 0 )

{

        gl.Rescale();

}
```

8.2.5 複数軸

次のサンプルコードは、1 つのグラフレイヤ内の 4 つの軸ー左、下、右、上の表示/非表示をセットし、フォーマットをセットする方法を示します。

```
#include <..¥Originlab¥graph_utils.h> // AXIS_* で必要

GraphLayer gl = Project.ActiveLayer();



// 全ての軸とラベルを表示。0 または 1, 1 は表示。

vector<int> vnAxes(4), vnLabels(4), vnTitles(4);

vnAxes[AXIS_BOTTOM] = 1;

vnAxes[AXIS_LEFT] = 1;

vnAxes[AXIS_TOP] = 1;

vnAxes[AXIS_RIGHT] = 1;

vnLabels = vnAxes;
```

```
// 左と下の軸の軸タイトルを表示。0 または 1, 1 は表示。

vnTitles[AXIS_BOTTOM] = 1;

vnTitles[AXIS_LEFT] = 1;

vnTitles[AXIS_TOP] = 0;

vnTitles[AXIS_RIGHT] = 0;



// すべての軸の主刻みと副刻みを内側にセット

// 他のオプションについては graph_utils.h の TICK_*項目を参照

vector<int> vnMajorTicks(4), vnMinorTicks(4);

vnMajorTicks[AXIS_BOTTOM] = TICK_IN;

vnMajorTicks[AXIS_LEFT] = TICK_IN;

vnMajorTicks[AXIS_TOP] = TICK_IN;

vnMajorTicks[AXIS_RIGHT] = TICK_IN;

vnMinorTicks = vnMajorTicks;



gl_smart_show_object(gl, vnAxes, vnLabels, vnTitles, vnMajorTicks, vnMinorTicks);
```

8.2.6　複数区分グラフ (X 軸を共有した複数レイヤ)

次のサンプルは、1 つのグラフページに複数のグラフレイヤを構築し、すべてのレイヤの X 軸を共有し、XY データセットをワークシートデータから各グラフレイヤにプロットする方法を示します。

次のコードをコンパイルする前に、現在のワークスペースに graph_utils.c をビルドするために、次のコマンドを実行する必要があります。

```
run.LoadOC(Originlab¥graph_utils.c, 16);
```

次の Origin C コードをコンパイルします。実行する前に、1 つの X 列と少なくとも 2 つの Y 列を持つ Book1 というワークブックがあることを確認します。

```
#include <..¥Originlab¥graph_utils.h> // page_add_layer 関数に対して必要

// Book1 からデータ範囲を構築

WorksheetPage wksPage("Book1");

Worksheet wks = wksPage.Layers(0); // Book1 の最初のワークシートを取得

DataRange dr;

dr.Add(wks, 0, "X"); // X データを第 1 列
```

```
dr.Add(wks, 1, "Y", -1); // 最後のＹデータを第２列

// Ｙの数を取得
DWORD dwRules = DRR_GET_DEPENDENT | DRR_NO_FACTORS;
int nNumYs = dr.GetNumData(dwRules);

// 右軸を持つレイヤを追加し、最初のレイヤにリンク
GraphPage gp;
gp.Create("Origin");
while ( gp.Layers.Count() < nNumYs )
{
        page_add_layer(gp, false, false, false, true,
                ADD_LAYER_INIT_SIZE_POS_MOVE_OFFSET, false, 0, LINK_STRAIGHT);
}

// ループして各 XY データからグラフレイヤにプロットを追加
foreach(GraphLayer gl in gp.Layers)
{
        int nLayerIndex = gl.GetIndex();

        // dr から部分 XY 範囲を取得
        DataRange drOne;
        dr.GetSubRange(drOne, dwRules, nLayerIndex);

        // １つの XY 範囲をグラフレイヤにプロット
        int nPlot = gl.AddPlot(drOne, IDM_PLOT_LINE);
        if( nPlot >= 0 )
        {
```

```
                DataPlot dp = gl.DataPlots(nPlot);
                dp.SetColor(nLayerIndex); // データプロットを異なる色にセット


                //  右Y軸の軸刻みと軸ラベルの色を自動にセット
                gl.YAxis.AxisObjects(AXISOBJPOS_AXIS_SECOND).RightTicks.Color.nVal =
                gl.YAxis.AxisObjects(AXISOBJPOS_LABEL_SECOND).RightLabels.Color.nVal =
                INDEX_COLOR_AUTOMATIC;


                gl.Rescale();
        }
}
```

8.3 データプロットを編集する

8.3.1 データマーカーを追加する

Origin C は、データマーカを編集するのに次のメソッドをサポートします。

- **DataPlot::AddDataMarkers** は、選択した部分範囲にデータプロットのデータマーカを追加します。
- **DataPlot::SetDataMarkers** は、現在のデータマーカの位置を変更します。
- **DataPlot::GetDataMarkers** は、すべての既存のデータプロットを取得します。
- **DataPlot::RemoveDataMarker** は、指定したデータマーカを削除します。

次のコードは、アクティブグラフに 2 つのデータマーカを追加する方法を示しています。

```
GraphLayer gl = Project.ActiveLayer();
DataPlot dp = gl.DataPlots();


// データマーカのインデックス
vector<int> vnBegin = {0, 9};
vector<int> vnEnd = {4, 14};


// 2つのデータマーカを追加
```

```
int nRet = dp.AddDataMarkers(vnBegin, vnEnd);

if( 0 == nRet )

{

        out_str("Add data marker successfully.");

}
```

以下のコードは、現在のデータマーカの位置を変更します。

```
GraphLayer gl = Project.ActiveLayer();

DataPlot dp = gl.DataPlots();



// データマーカのインデックス

vector<int> vnBegin = {11, 2};

vector<int> vnEnd = {19, 5};

vector<int> vnIndices = {1, 0};



// 2 つのデータマーカを追加

int nRet = dp.SetDataMarkers(vnBegin, vnEnd, vnIndices);

if( 0 == nRet )

{

        out_str("Set data marker successfully.");

        gl.GetPage().Refresh();

}
```

8.3.2 色をセットする

次のサンプルコードは、データプロットの色をセットする方法を示します。

```
GraphLayer gl = Project.ActiveLayer();

DataPlot dp = gl.DataPlots(0);



bool bRepaint = true;

dp.SetColor(SYSCOLOR_GREEN, bRepaint);
```

8.3.3 フォーマットツリーの取得

OriginObject::GetFormat と *OriginObject::ApplyFormat* は、Origin オブジェクトフォーマットを取得するのに使われます。次のフォーマットの取得、セット、コピーのメカニズムは、OriginObject の基底クラスから派生するクラスを持つすべての Origin オブジェクトに対して使うことができます。(参照:クラスの階層)例えば、Origin オブジェクトは、*DataPlot* クラス, *Worksheet* クラス, *WorksheetPage* クラス, *MatrixLayer* クラス, *MatrixPage* クラス, *GraphLayer* クラス, *GraphPage* クラスのオブジェクトにすることができます。

DataPlot クラスは、*DataObjectBase* クラスから派生され、*DataObjectBase* クラスは、*OriginObject* クラスがら派生されるので、*DataPlot::GetFormat* を呼び出して、フォーマットツリー構造を取得できます。

次のコードを使って、フォーマットツリー構造を見る 2 つの方法があります。

- 次のコードで、*GetFormat* の行にブレークポイントを設定し、1 つのデータをアクティブにし、コードを実行し、F10 を押し(ステップオーバー)、*GetFormat* 行を実行し、コードビルダのローカル変数ウィンドウで、**tr** 変数のフォーマットツリーの詳細を見ます。(Alt+4 を押し、ローカル変数ウィンドウを開いたり、閉じたりできます。)
- 最後の行 *out_tree(tr);,* を使って、フォーマットツリーを出力します。

```
GraphLayer gl = Project.ActiveLayer();

DataPlot dp = gl.DataPlots(-1); // アクティブデータプロットを取得



// 異なるプロットタイプ(例えば、折れ線、ボックスチャート...) は、

// フォーマットツリーの異なる構造を持ちます。

Tree tr;



// ツリー構造の詳細を表示するフォーマットツリーを取得

tr = dp.GetFormat(FPB_ALL, FOB_ALL, true, true);



out_tree(tr); // フォーマットツリーを取得
```

8.3.4 折れ線グラフのフォーマットをセットする

```
GraphLayer gl = Project.ActiveLayer();

DataPlot dp = gl.DataPlots(-1); //  アクティブデータプロットを取得



// 折れ線プロットのフォーマットをセット

// Note:フォーマットツリーの構造を取得するには前のセクションを参照

Tree tr;
```

```
tr.Root.Line.Connect.nVal = 2; // 2 点のセグメントに対しては 2

tr.Root.Line.Color.nVal = RGB2OCOLOR(RGB(100, 100, 220));

tr.Root.Line.Width.dVal = 1.5;



if( 0 == dp.UpdateThemeIDs(tr.Root) )

{

    bool bRet = dp.ApplyFormat(tr, true, true);

}
```

8.3.5 あるデータプロットから別のプロットにフォーマットをコピーする

テーマファイルでフォーマットをコピーする

データプロットからテーマファイルにフォーマットツリーを取得し保存すると、テーマファイルをツリーにロードし、フォーマットツリーを別のデータプロットに適用できます。

```
// Graph1 からテーマファイルにプロット設定を保存

GraphPage gpSource("Graph1");

GraphLayer glSource = gpSource.Layers(0);

DataPlot dpSource = glSource.DataPlots(0);



Tree tr;

tr = dpSource.GetFormat(FPB_ALL, FOB_ALL, true, true);

string strTheme = GetAppPath(false) + "plotsettings.XML";

tr.Save(strTheme);



// テーマファイルからプロット設定をツリーにロードし、

// ツリーからデータプロットオブジェクトにフォーマットを適用

GraphPage gpDest("Graph2");

GraphLayer glDest = gpDest.Layers(0);

DataPlot dpDest = glDest.DataPlots(0);
```

```
Tree tr2;

tr2.Load(strTheme);

dpDest.ApplyFormat(tr2, true, true);
```

ツリーでフォーマットをコピーする

あるデータプロットからのプロット設定をツリーに取得すると、このツリーから別のプロットオブジェクトに設定を適用できます。

```
GraphPage gpSource("Graph1");

GraphLayer glSource = gpSource.Layers(0);

DataPlot dpSource = glSource.DataPlots(0);



GraphPage gpDest("Graph2");

GraphLayer glDest = gpDest.Layers(0);

DataPlot dpDest = glDest.DataPlots(0);



// 元のデータプロットからフォーマットを取得

Tree tr;

tr = dpSource.GetFormat(FPB_ALL, FOB_ALL, true, true);



// フォーマットを別のデータプロットに適用

dpDest.ApplyFormat(tr, true, true);
```

8.3.6　散布図のフォーマットをセットする

```
GraphLayer gl = Project.ActiveLayer();

DataPlot dp = gl.DataPlots(-1); // アクティブデータプロットを取得



// シンボルフォーマットをセット

Tree tr;

tr.Root.Symbol.Size.nVal = 12; // シンボルのサイズ

tr.Root.Symbol.Shape.nVal = 1; // 円
```

```
tr.Root.Symbol.Interior.nVal = 1; // 内部

tr.Root.Symbol.EdgeColor.nVal = SYSCOLOR_RED;

tr.Root.Symbol.FillColor.nVal = SYSCOLOR_BLUE;



// ドロップラインの表示

tr.Root.DropLines.Vertical.nVal = 1;

tr.Root.DropLines.VerticalColor.nVal = SYSCOLOR_LTGRAY;

tr.Root.DropLines.VerticalStyle.nVal = 1;

tr.Root.DropLines.VerticalWidth.nVal = 1.5;



if( 0 == dp.UpdateThemeIDs(tr.Root) )

{

    bool bRet = dp.ApplyFormat(tr, true, true);

}
```

8.3.7　グループ化した線＋シンボルのフォーマットを設定する

Origin C を使って、グループ化したプロットに対するフォーマットをセットします。同じ操作を、作図の詳細ダイアログで、グループタブで行うことができます。フォーマットには線の色、シンボルタイプ、シンボルの内部、線種が含まれます。

次のサンプルは、線＋シンボルグラフのフォーマットをセットする方法を示しています。このグループには 4 つのデータプロットがあるものとします。

```
GraphLayer gl = Project.ActiveLayer();

GroupPlot gplot = gl.Groups(0); // レイヤ内の最初のグループ



// Nester はグループ内でネストするオブジェクトタイプの配列

// グループ内で周期的にネストする 4 種類の設定

vector<int> vNester(3);

vNester[0] = 0;  // グループ内で周期的に線の色を変更

vNester[1] = 3;  // グループ内で周期的にシンボルを変更

vNester[2] = 8;  // グループ内で周期的にシンボル内部を変更

gplot.Increment.Nester.nVals = vNester;  // グループプロットの Nester
```

```
// フォーマット設定を 4 つのプロットへの vector に配置
vector<int> vLineColor = {SYSCOLOR_BLUE, SYSCOLOR_OLIVE, SYSCOLOR_RED,
                SYSCOLOR_CYAN};
vector<int> vSymbolShape = {1, 3, 5, 8};
vector<int> vSymbolInterior = {1, 2, 5, 0};

Tree tr;
tr.Root.Increment.LineColor.nVals = vLineColor;  // 線の色をテーマツリーにセット
tr.Root.Increment.Shape.nVals = vSymbolShape;  // シンボルの形状をテーマツリーにセット
// シンボルの内部をテーマツリーにセット
tr.Root.Increment.SymbolInterior.nVals = vSymbolInterior;

if(0 == gplot.UpdateThemeIDs(tr.Root) )
{
        bool bb = gplot.ApplyFormat(tr, true, true);    // テーマツリーを適用
}
```

8.3.8　カラーマップ設定を設定する

DataPlot クラスは、カラーマップの設定をするための 2 つのオーバーロードメソッド持ちます。

- DataPlot::SetColormap(const vector<double> & vz, BOOL bLogScale = FALSE) は、Z 値レベルとスケールタイプ（対数かそうでないか）の設定だけに使用されます。**vz** 引数内の値は Z 値です。
- DataPlot::SetColormap(TreeNode& trColormap) は、すべてのカラーマップ設定に使用されます。たとえば、Z 値、色、線のフォーマット、テキストラベルのフォーマット等の設定をします。

次のサンプルでは、等高線図の Z カラーマップの設定方法を示します。

```
GraphLayer gl = Project.ActiveLayer();
DataPlot dp = gl.DataPlots(0);

// Z レベルの元のカラーマップを取得
```

```
vector vZs;
BOOL bLogScale = FALSE;
BOOL bRet = dp.GetColormap(vZs, bLogScale);
int nLevels = vZs.GetSize();

// Z レベルの vector を減少させ、DataPlot に戻してセット
double min, max;
vZs.GetMinMax(min, max);
double dChangeVal = fabs(max - min) * 0.2;
bool bIncrease = true;
if( !bIncrease )
        dChangeVal = 0 - dChangeVal;

min = min - dChangeVal;
max = max - dChangeVal;
double inc = (max - min) / nLevels;
vZs.Data(min, max, inc);

dp.SetColormap(vZs);
```

次のサンプルでは、カラーマップ Z 値レベルを Log10 スケールタイプでセットする方法を示しています。

```
bool plot_matrix(LPCSTR lpszMatPage, LPCSTR lpcszGraphTemplate = "contour"
                , int nPlotID = IDM_PLOT_CONTOUR)
{
        // 特定行列ページから行列オブジェクトを取得
        MatrixPage matPage = Project.MatrixPages(lpszMatPage);
        if( !matPage )
        {
                out_str("Invalid matrix page");
```

```
        return false;
}
// 行列ページのアクティブシートを取得
MatrixLayer ml = matPage.Layers(-1);
// 行列シート内のアクティブ行列オブジェクトを取得
MatrixObject mobj = ml.MatrixObjects(-1);

// テンプレートで非表示のグラフページを作成し、プロットを追加
// 非表示として、描画しない
GraphPage gp;
gp.Create(lpcszGraphTemplate, CREATE_HIDDEN);
GraphLayer glay = gp.Layers();

int nPlot = glay.AddPlot(mobj, nPlotID);
if(nPlot < 0)
{
        out_str("fail to add data plot to graph");
    return false;
}
glay.Rescale(); // XY 軸を再スケール

// Z レベルベクトルを構築
int nNewLevels = 4;
double min = 0.1, max = 100000.;
double step = (log10(max) - log10(min)) / (nNewLevels - 1);

vector vLevels;
vLevels.SetSize(nNewLevels);
```

```
        vLevels.Data(log10(min), log10(max), step);
        vLevels = 10^vLevels;

        // パーセントで Z 値をセット。実際の Z 値ではないことに注意
        // 最初の値は 0、最後の値は < 100 である必要がある
        vLevels = 100*(vLevels - min)/(max - min);

        Tree tr;
        tr.ColorMap.Details.Levels.dVals = vLevels;
        tr.ColorMap.ScaleType.nVal = 1; // 1 は log10
        tr.ColorMap.Min.dVal = min;
        tr.ColorMap.Max.dVal = max;

        DataPlot dp = glay.DataPlots(nPlot);
        bool bRet = dp.SetColormap(tr);
        if( !bRet )
        {
                out_str("fail to set colormap");
                return false;
        }

        gp.Label = "Plot created using template:" + (string)lpcszGraphTemplate;
        gp.TitleShow = WIN_TITLE_SHOW_BOTH;
        gp.SetShow(); // 用意できたら表示

        return true;
}
```

上述の *plot_matrix* 関数を *coutour* テンプレートとともに呼び出し、*IDM_PLOT_CONTOUR* プロット id で等高線図を作成してカラーマップをセットします。

```
void plot_contour_ex(LPCSTR lpcszMatPage)

{

    plot_matrix(lpcszMatPage, "contour", IDM_PLOT_CONTOUR);

}
```

上述の *plot_matrix* 関数を *image* テンプレートと共に呼び出し、*IDM_PLOT_MATRIX_IMAGE* プロット id でイメージグラフを作成してカラーマップをセットします。

```
void plot_image_ex(LPCSTR lpcszMatPage)

{

    plot_matrix(lpcszMatPage, "image", IDM_PLOT_MATRIX_IMAGE);

}
```

次のサンプルでは、塗りつぶし色を削除し、等高線図の線の色、スタイル、幅、テキストラベルをセットします。

```
GraphLayer gl = Project.ActiveLayer();

DataPlot dp = gl.DataPlots(0);



Tree tr;

dp.GetColormap(tr);



// 塗りつぶし色削除

tr.ColorFillControl.nVal = 0;



// 線の色を設定

vector<int> vnLineColors;

vnLineColors = tr.Details.LineColors.nVals;

int nLevels = vnLineColors.GetSize();

vnLineColors.Data(1, nLevels, 1);

tr.Details.LineColors.nVals = vnLineColors;
```

```
// 全ての線のスタイルを破線にセット
vector<int> vnLineStyles(vnLineColors.GetSize());
vnLineStyles = 1;
tr.Details.LineStyles.nVals = vnLineStyles;

// 全ての線の幅をセット
vector vdLineWidths(vnLineColors.GetSize());
vdLineWidths = 3;
tr.Details.LineWidths.dVals = vdLineWidths;

// ラベルの表示/非表示。最初の 2 つ以外を表示
vector<int> vnLabels(vnLineColors.GetSize());
vnLabels = 1;
vnLabels[0] = 0;
vnLabels[1] = 0;
tr.Details.Labels.nVals = vnLabels;

// グラフの背景の設定
dp.SetColormap(tr);
```

このサンプルでは、等高線グラフのテキストラベルのフォーマット(例:色、サイズ、太字、斜体)をセットする方法を示します。

```
GraphLayer gl = Project.ActiveLayer();
DataPlot dp = gl.DataPlots(0);

// カラーマッププロットの関連オブジェクトのすべてのプロパティを取得
Tree tr;
tr = dp.GetFormat(FPB_ALL, FOB_ALL, true, true);

// すべてのラベルを表示
```

```
vector<int> vnLabels;

vnLabels = tr.Root.ColorMap.Details.Labels.nVals;

vnLabels = 1;// 0 は非表示、1 は表示

tr.Root.ColorMap.Details.Labels.nVals = vnLabels;



// ラベルに対して数値フォーマットをセット

tr.Root.NumericFormats.Format.nVal = 0; // 十進法

tr.Root.NumericFormats.DigitsControl.nVal = 0;

tr.Root.NumericFormats.SignificantDigits.nVal = 5;//小数点以下桁数

tr.Root.NumericFormats.Prefix.strVal = "_";

tr.Root.NumericFormats.Suffix.strVal = "Label";

tr.Root.NumericFormats.MinArea.nVal = 5; // ラベル Criteria - Min Area(%)



// ラベルに対してテキストフォーマットをセット

tr.Root.Labels.Color.nVal = SYSCOLOR_BLUE;

//FontFaceIndex_to_DWORD は、GUI インデックスから DWORD 実数値に変換するのに使用

tr.Root.Labels.Face.nVal = FontFaceIndex_to_DWORD(2);// GUI 内に 3 番目のフォントを選択

tr.Root.Labels.Size.nVal = 20;

tr.Root.Labels.WhiteOut.nVal = 1;

tr.Root.Labels.Bold.nVal = 1;

tr.Root.Labels.Italic.nVal = 1;

tr.Root.Labels.Underline.nVal = 1;



if(0 == dp.UpdateThemeIDs(tr.Root) )

        dp.ApplyFormat(tr, true, true);
```

8.4 レイヤを管理する

8.4.1 区分プロットを作成する

区分プロットを作成する

次のサンプルは、2 列 3 行で配置した 6 つのレイヤを持つ新しいグラフを作成します。この関数は、どのウィンドウがアクティブかに関係なく実行できます。

```
GraphPage gp;

gp.Create("Origin");



while(gp.Layers.Count() < 6)

{

        gp.AddLayer();

}



graph_arrange_layers(gp, 3, 2);
```

6 区分グラフを作成しプロット

次のサンプルは、新しいワークブックにいくつかデータをインポートし、6 つのレイヤを持つ新しいグラフウィンドウを作成し、2 列 3 行で配置し、各レイヤ(区分)をループして、インポートしたデータをプロットします。

```
// ワークシートにデータファイルをインポート

ASCIMP          ai;

Worksheet       wks;

string  strDataFile = GetOpenBox(FDLOG_ASCII, GetAppPath(true));

if(AscImpReadFileStruct(strDataFile,&ai) == 0)

{

        wks.Create("Origin");

        wks.ImportASCII(strDataFile, ai);

}



// ワークシートからグラフの各レイヤに XY データを追加

GraphPage gp("Graph1"); // グラフは上記で作成した 3x2 区分レイヤを持つ
```

```
int index = 0;

foreach(GraphLayer gl in gp.Layers)

{

        DataRange dr;

        dr.Add(wks, 0, "X");

        dr.Add(wks, index+1, "Y");



        if( gl.AddPlot(dr, IDM_PLOT_LINE) >= 0 )

                gl.Rescale();



        index++;

}
```

8.4.2 グラフウィンドウにレイヤを追加する

次のサンプルは、独立した右 Y 軸スケールを追加します。右 Y 軸のみを表示する新しいレイヤを追加します。これは寸法がリンクされ、レイヤが追加された時に、X 軸が現在のアクティブレイヤにリンクされます。新しく追加したレイヤがアクティブレイヤになります。

次のコードをコンパイルする前に、現在のワークスペースに graph_utils.c を追加する必要があります。これには、Labtalk コマンド"Run.LoadOC(Originlab\graph_utils.c)"を使います。

```
#include <..¥Originlab¥graph_utils.h>// page_add_layer 関数に必要



GraphLayer gl = Project.ActiveLayer();

GraphPage gp = gl.GetPage();



bool bBottom = false, bLeft = false, bTop = false, bRight = true;

int nLinkTo = gl.GetIndex(); // アクティブレイヤにレイヤリンクを新しく追加

bool bActivateNewLayer = true;



int        nLayerIndex = page_add_layer(gp, bBottom, bLeft, bTop, bRight,

ADD_LAYER_INIT_SIZE_POS_SAME_AS_PREVIOUS, bActivateNewLayer, nLinkTo);
```

8.4.3 アクティブなレイヤ以外を非表示にする

```
GraphPage gp("Graph1");

if( gp )

{

        GraphLayer glActive = gp.Layers(-1); // -1 はアクティブレイヤを取得



        foreach(GraphLayer gl in gp.Layers)

        {

                if( gl.GetIndex() != glActive.GetIndex() )

                        gl.Show(false);

        }

}
```

8.4.4 レイヤを配置する

次のサンプルは、アクティブグラフに既存レイヤを 2 行 3 列で配置します。アクティブグラフに 6 つのレイヤが無くても、新しいレイヤは追加されません。存在しているレイヤのみが配置されます。

```
GraphLayer gl = Project.ActiveLayer();

GraphPage gp = gl.GetPage();



int nRows = 3, nCols = 2;

graph_arrange_layers(gp, nRows, nCols);
```

8.4.5 レイヤを移動する

次のサンプルは、アクティブグラフウィンドウのすべてのレイヤを左に整列し、ページの左側から 15%の位置にセットします。

```
GraphLayer gl = Project.ActiveLayer();

GraphPage gp = gl.GetPage();



int nRows = gp.Layers.Count();

int nCols = 1;
```

```
stLayersGridFormat stFormat;

stFormat.nXGap = 0; // レイヤの X 方向の余白

stFormat.nYGap = 5; // レイヤの Y 方向の余白

stFormat.nLeftMg = 15; // 左余白

stFormat.nRightMg = 10;

stFormat.nTopMg = 10;

stFormat.nBottomMg = 10;



page_arrange_layers(gp, nRows, nCols, &stFormat);
```

8.4.6　レイヤサイズの変更

次のサンプルは、現在のレイヤのサイズを変更し、幅と高さを元のサイズの半分にします。

次のコードをコンパイルする前に、現在のワークスペースに graph_utils.c を追加する必要があります。これには、Labtalk コマンド"Run.LoadOC(Originlab\graph_utils.c)"を使います。

```
#include <..¥Originlab¥graph_utils.h> // layer_set_size 関数に必要

GraphLayer gl = Project.ActiveLayer();



// グラフレイヤの元のサイズを取得

double dWidth, dHeight;

layer_get_size(gl, dWidth, dHeight);



// レイヤサイズ変更

dWidth /= 2;

dHeight /= 2;

layer_set_size(gl, dWidth, dHeight);
```

8.4.7　２つのレイヤを入れ替える

次のサンプルは、レイヤ１と２のページの位置をインデックス番号を使って入れ替えます。

次のコードをコンパイルする前に、現在のワークスペースに graph_utils.c を追加する必要があります。これには、Labtalk コマンド"Run.LoadOC(Originlab\graph_utils.c)"を使います。

```
#include <..¥Originlab¥graph_utils.h> // layer_swap_position 関数に必要

GraphPage gp("Graph1");
```

```
GraphLayer gl1 = gp.Layers(0);

GraphLayer gl2 = gp.Layers(1);

layer_swap_position(gl1, gl2);
```

次のサンプルは、Layer1 および Layer2 という名前のレイヤの位置を名前を入れ替えます。

```
GraphPage gp("Graph1");

GraphLayer gl1 = gp.Layers("Layer1");

GraphLayer gl2 = gp.Layers("Layer2");

layer_swap_position(gl1, gl2);
```

8.4.8 レイヤを整列する

次のサンプルは、アクティブグラフレイヤのレイヤ 1 とレイヤ 2 を下辺に合わせて整列します。

次のコードをコンパイルする前に、現在のワークスペースに graph_utils.c を追加する必要があります。これには、Labtalk コマンド"Run.LoadOC(Originlab\graph_utils.c)"を使います。

```
#include <..¥Originlab¥graph_utils.h> // layer_aligns 関数に必要

// アクティブグラフページを取得

GraphLayer gl = Project.ActiveLayer();

GraphPage gp = gl.GetPage();



GraphLayer gl1 = gp.Layers(0);

GraphLayer gl2 = gp.Layers(1);



// レイヤ 2 をレイヤ 1 に下辺で合わせる

layer_aligns(gl1, gl2, POS_BOTTOM);
```

8.4.9 レイヤをリンクする

次のサンプルは、アクティブグラフのすべてのレイヤの X 軸をレイヤ 1 の X 軸にリンクします。単位は、リンクしたレイヤの%にセットされます。

次のコードをコンパイルする前に、現在のワークスペースに graph_utils.c を追加する必要があります。これには、Labtalk コマンド"Run.LoadOC(Originlab\graph_utils.c)"を使います。

```
#include <..¥Originlab¥graph_utils.h> // layer_set_link 関数に必要

GraphLayer gl = Project.ActiveLayer();

GraphPage gp = gl.GetPage();
```

```
GraphLayer gl1 = gp.Layers(0); // Layer 1

foreach(GraphLayer glOne in gp.Layers)

{

        int nUnit = M_LINK; // レイヤの単位をリンクしたレイヤの%にセット

        if( glOne != gl1 )

                layer_set_link(glOne, gl1.GetIndex(), LINK_STRAIGHT, LINK_NONE, &nUnit);

}
```

8.4.10 レイヤ単位をセットする

```
int nUnit = M_PIXEL;

GraphLayer gl = Project.ActiveLayer();



// 現在の位置を取得

double dPos[TOTAL_POS];

gl.GetPosition(dPos);



// 位置を指定した単位に変換

gl.UnitsConvert(nUnit, dPos);



// 位置を単位でセット

gl.SetPosition(dPos, nUnit);
```

8.5 グラフィックオブジェクトの作成とアクセス

8.5.1 グラフィックオブジェクトの作成

テキストや四角形、線といったグラフィックオブジェクトを追加します。

次のサンプルでは、アクティブなグラフに四角形を追加する方法を紹介します。他のグラフオブジェクトタイプについては、oc_const.h ファイル内の GROT_* (例:GROT_TEXT, GROT_LINE, GROT_POLYGON) を確認してください。

```
GraphLayer gl = Project.ActiveLayer();

string strName = "MyRect";

GraphObject goRect = gl.CreateGraphObject(GROT_RECT, strName);
```

現在のグラフウィンドウにテキストラベルを追加します。

```
GraphLayer gl = Project.ActiveLayer();

GraphObject go = gl.CreateGraphObject(GROT_TEXT, "MyText");

go.Text = "This is a test";
```

以下のサンプルでは、グラフに矢印を追加する方法を示します。矢印のオブジェクトタイプは、GROT_LINE で、線と同じです。線と矢印ともに、データポイントの数は 2 つ必要です。

```
GraphPage gp;

gp.Create();

GraphLayer gl = gp.Layers();



string strName = "MyArrow"; // グラフオブジェクトの名前

GraphObject go = gl.CreateGraphObject(GROT_LINE, strName);



go.Attach = 2; // 接続先はレイヤとスケール



Tree tr;

tr.Root.Dimension.Units.nVal = 5; // 単位はスケールにセット



// スケール値で位置をセット

vector vx = {2, 6};

vector vy = {6, 2};

tr.Root.Data.X.dVals = vx;

tr.Root.Data.Y.dVals = vy;



tr.Root.Arrow.Begin.Style.nVal = 0;

tr.Root.Arrow.End.Style.nVal = 1;



if( 0 == go.UpdateThemeIDs(tr.Root) )
```

```
{
        go.ApplyFormat(tr, true, true);
}
```

次のサンプルでは、グラフに曲線矢印を追加する方法を示します。曲線矢印は、データポイントの数は４つ必要です。

```
GraphPage gp;
gp.Create();
GraphLayer gl = gp.Layers();

string strName = "MyArrow"; // グラフオブジェクトの名前
GraphObject go = gl.CreateGraphObject(GROT_LINE4, strName);

go.Attach = 2; // 接続先はレイヤとスケール

Tree tr;
tr.Root.Dimension.Units.nVal = 5; // 単位をスケールにセット

// スケール値で位置をセット
vector vx = {2, 4, 6, 5};
vector vy = {7, 6.9, 6.8, 2};
tr.Root.Data.X.dVals = vx;
tr.Root.Data.Y.dVals = vy;
tr.Root.Arrow.Begin.Style.nVal = 0;
tr.Root.Arrow.End.Style.nVal = 1;

if( 0 == go.UpdateThemeIDs(tr.Root) )
{
        go.ApplyFormat(tr, true, true);
}
```

8.5.2 プロパティ設定

テキストフォントや、色、線の幅などのグラフィカルオブジェクトのプロパティを設定します。

```
// グラフオブジェクトの色とフォントを設定

GraphLayer gl = Project.ActiveLayer();

GraphObject goText = gl.GraphObjects("Text");

goText.Text = "This is a test";

goText.Attach = 2; // レイヤスケールに接続



Tree tr;

tr.Root.Color.nVal = SYSCOLOR_RED; // テキストの色

tr.Root.Font.Bold.nVal = 1;

tr.Root.Font.Italic.nVal = 1;

tr.Root.Font.Underline.nVal = 1;

tr.Root.Font.Size.nVal = 30; // テキストのフォントサイズ



if( 0 == goText.UpdateThemeIDs(tr.Root) )

{

        bool bRet = goText.ApplyFormat(tr, true, true);

}
```

8.5.3 位置とサイズを設定

```
GraphLayer gl = Project.ActiveLayer();

GraphObject go = gl.GraphObjects("Rect");

go.Attach = 2; // レイヤスケールに接続



// テキストオブジェクトをレイヤの左上に移動

Tree tr;

tr.Root.Dimension.Units.nVal = UNITS_SCALE;

tr.Root.Dimension.Left.dVal = gl.X.From; // 左
```

```
tr.Root.Dimension.Top.dVal = gl.Y.To/2; // 上

tr.Root.Dimension.Width.dVal = (gl.X.To - gl.X.From)/2; // 幅

tr.Root.Dimension.Height.dVal = (gl.Y.To - gl.Y.From)/2; // 高さ



if( 0 == go.UpdateThemeIDs(tr.Root) )

{

        bool bRet = go.ApplyFormat(tr, true, true);

}
```

8.5.4　接続先プロパティを更新

接続先には、ページ、レイヤ枠、レイヤスケールの 3 つの選択肢があります。

```
// グラフオブジェクトに異なるオブジェクトを接続

// 0 はレイヤ。レイヤを移動するとグラフオブジェクトも一緒に移動

// 1 はページ。レイヤを移動してもグラフオブジェクトに影響はない

// 2 はレイヤスケール。スケールを変更するとグラフオブジェクトの位置が

// 対応して移動

go.Attach = 2;
```

8.5.5　無効なプロパティの取得と設定

```
// 移動可能、選択可能といった無効なプロパティを確認

Tree tr;

tr = go.GetFormat(FPB_OTHER, FOB_ALL, true, true);

DWORD   dwStats = tr.Root.States.nVal;



// 垂直と水平移動を確認

// プロパティの詳細は、oc_const.h ファイルの GOC_* を確認

if( (dwStats & GOC_NO_VMOVE) && (dwStats & GOC_NO_HMOVE) )

{

        out_str("This graph object cannot be move");

}
```

8.5.6 プログラム制御

```
// 1.線を追加
GraphLayer gl = Project.ActiveLayer();
GraphObject go = gl.CreateGraphObject(GROT_LINE);
go.Attach = 2; // 接続先をレイヤスケールにセット
go.X = 5; // 初期位置 X = 5 にセット

// 2.線図のプロパティをセット
Tree tr;
tr.Root.Direction.nVal = 2; // 1 は水平、2 は垂直
tr.Root.Span.nVal = 1; // レイヤまで間隔
tr.Root.Color.nVal = SYSCOLOR_RED; // 線の色

if( 0 == go.UpdateThemeIDs(tr.Root) )
{
        go.ApplyFormat(tr, true, true);
}

// 3.イベントモードと LT スクリプトをセット
// 線移動すると線の位置の X 値を印字
Tree trEvent;
trEvent.Root.Event.nVal = GRCT_MOVE;// 詳細は、oc_const.h の GRCT_*
trEvent.Root.Script.strVal = "type -a $(this.X)";

if( 0 == go.UpdateThemeIDs(trEvent.Root) )
{
        go.ApplyFormat(trEvent, true, true);
}
```

8.5.7　凡例の更新

凡例は、グラフウィンドウ内にある"Legend"という名前のグラフィックオブジェクトです。データプロットを追加/削除後、**legend_update** 関数を使用して、現在データプロットに基づいた凡例に更新することが可能です。

```
// 凡例の更新の簡単な使用方法

// Origin C ヘルプでは、この関数を検索して引数や

// 使用例について確認可能

legend_update(gl); // gl は、GraphLayer オブジェクト
```

8.5.8　グラフに表オブジェクトを追加

```
// 1.表テンプレートとともにワークシーとを作成

Worksheet wks;

wks.Create("Table", CREATE_HIDDEN);

WorksheetPage wksPage = wks.GetPage();



// 2.表のサイズを取得し、テキストを埋める

wks.SetSize(3, 2);

wks.SetCell(0, 0, "1");

wks.SetCell(0, 1, "Layer 1");



wks.SetCell(1, 0, "2");

wks.SetCell(1, 1, "Layer 2");



wks.SetCell(2, 0, "3");

wks.SetCell(2, 1, "Layer 3");



//3.グラフへのリンクとして表を追加

GraphLayer gl = Project.ActiveLayer();

GraphObject grTable = gl.CreateLinkTable(wksPage.GetName(), wks);
```

9 データ操作

9.1 数値データ

このセクションでは、Origin C で数値データを操作するサンプルを扱います。数値データは次のデータタイプの変数に保存することができます。

1. double
2. Integer
3. vector
4. matrix

数値データと文字列は、ツリーのノードに保存でき、提供されたノードは上記のデータ型の 1 つです。

9.1.1 欠損値

数値データが重要であるのと同様に、欠損値を表すことができることも重要です。Origin C は、値が欠損値であるかどうかを比較する NANUM マクロを定義します。欠損値は、double 型のデータ型のみをサポートしています。

```
double d = NANUM;

if( NANUM == d )

    out_str("The value is a missing value.");
```

Origin C は、値が欠損値であるかどうかをチェックする **is_missing_value** 関数も提供しています。

```
if( is_missing_value(d) )

    out_str("The value is a missing value.");
```

9.1.2 精度と比較

次のサンプルコードでは、*prec* と *round* 関数を使って、double 型の数値データの精度を制御します。*is_equal* 関数を使って、double 型の数値データの 2 つを比較します。

```
double dVal = PI; // PI は 3.14159265358979323846 と定義

// double 型の値を有効桁数 6 ケタに変換

int nSignificantDigits = 6;

printf("%f¥n", prec(dVal, nSignificantDigits));



// double 型の値を 2 ケタのみにする

uint nDecimalPlaces = 2;
```

```
double dd = round(dVal, nDecimalPlaces);

printf("%f¥n", dd);



// 2 つの double 型の値を比較

if( is_equal(dd, 3.14) )

{

        out_str("equal¥n");

}

else

{

        out_str("not equal¥n");

}
```

9.1.3 数値を文字列に変換する

```
// int 型の数値を文字列に割り当て

string str = 10;

out_str(str);



int nn = 0;

str = nn;

out_str(str);



// double 型の数値を文字列に割り当て

double dd = PI;

str = ftoa(dd, "*"); // オプションダイアログで Origin のグローバル設定に"*"を使う

out_str(str);



str = ftoa(dd, "*8"); // 有効桁数 8 ケタに"*8"を使用

out_str(str);
```

9.1.4　ベクトル

```
// 基本データ型、例えば double, int, string, complex の
//  1 次配列
vector vx, vy;

int nMax = 10;
vx.Data(1, nMax, 1); // 1 から 10 まで増分 1 で値を vx に割り当て
vy.SetSize(nMax); // サイズ(10)を vy にセット

for(int nn = 0; nn < nMax; nn++)
{
        vy[nn] = rnd(); // 乱数を vy の各項目に割り当て
        printf("index = %d, x = %g, y = %g¥n", nn+1, vx[nn], vy[nn]);
}

// ワークシート内のデータにアクセス
Worksheet wks = Project.ActiveLayer();
Column col(wks, 0);

vector& vec = col.GetDataObject();
vec = vec * 0.1; // vec 内の各データに 0.1 を乗算

vec = sin(vec);     // vec 内のデータの sin を求める
```

9.1.5　行列

```
// 基本データ型、例えば double, int, complex の 2 次配列
// string は含まれない
matrix mat(5, 6);
```

```
for(int ii = 0; ii < 5; ii++)
{
        for(int jj = 0; jj < 6; jj++)
        {
                mat[ii][jj] = ii + jj;
                printf("%g¥t", mat[ii][jj]);
        }
        printf("¥n"); // 新しいライン
}

// 行列ウィンドウ内のデータにアクセス
MatrixLayer ml = Project.ActiveLayer();
MatrixObject mo = ml.MatrixObjects(0);

matrix& mat = mo.GetDataObject();
mat = mat + 0.1; // 行列の各データに 0.1 を加算
```

9.1.6 TreeNode

Origin C の TreeNode クラスは、複数レベルのツリーを構築し、ツリーを横断し、ツリーノードの値/属性にアクセスするいくつかのメソッドを提供します。

```
Tree tr;

// ツリーノードの値にアクセス
TreeNode trName = tr.AddNode("Name");
trName.strVal = "Jane";
tr.UserID.nVal = 10;
vector<string> vsBooks = {"C++", "MFC"};
tr.Books.strVals = vsBooks;

out_tree(tr); // ツリーを出力
```

9.1.7 複素数

```
complex cc(1.5, 2.2);



cc.m_re = cc.m_re +1;

cc.m_im = cc.m_im * 0.1;



out_complex("cc = ", cc); // cc = 2.500000+0.220000i



// 複素数データセットにアクセス

Worksheet wks = Project.ActiveLayer();

Column col(wks, 1);

if( FSI_COMPLEX == col.GetInternalDataType() )

{

        vector<complex>& vcc = col.GetDataObject();

        vcc[0] = 0.5 + 3.6i;

}



// 複素数行列にアクセス

MatrixLayer ml = Project.ActiveLayer();

MatrixObject mo = ml.MatrixObjects();



if( FSI_COMPLEX == mo.GetInternalDataType() )

{

        matrix<complex>& mat = mo.GetDataObject();

        mat[0][0] = 1 + 2.5i;

}
```

9.1.8 DataRange

DataRange クラスは、ワークシート、行列、グラフウィンドウでデータを取得したり、配置する広範なメカニズムを提供します。

ワークシート内のデータ範囲

ワークシートに対しては、データ範囲は、1 列、1 行、部分範囲、1 つのセル、ワークシート全体として列と行のインデックスを指定することができます。

```
// アクティブワークシートの 1 行目から 5 行目までのすべての列と行の

// データ範囲を構築

Worksheet wks = Project.ActiveLayer();

int r1 = 0, c1 = 0, r2 = 4, c2 = -1;



DataRange dr;

// 範囲名は"X", "Y"

//  "ED"(Y エラー), "Z"のようなわかりやすいものにデータ範囲は、従属または独立のデータ型に属さなければ

// デフォルトは "X"。

dr.Add("X", wks, r1, c1, r2, c2);
```

データ範囲から vector にデータを取得します。**DataRange::GetData** は、複数のオーバーロードしたメソッドをサポートします。例えば、

```
vector vData;

int index = 0; // 範囲インデックス

dr.GetData(&vData, index);
```

行列内のデータ範囲

行列ウィンドウに対しては、データ範囲は行列オブジェクトインデックスにすることができます。

```
MatrixLayer ml = Project.ActiveLayer();

DataRange dr;

int nMatrixObjectIndex = 0;

dr.Add(ml, nMatrixObjectIndex, "X");
```

データ範囲から matrix にデータを取得します。

```
matrix mat;

dr.GetData(mat);
```

グラフ内のデータ範囲

グラフウィンドウに対して、データ範囲はデータプロットの 1 つまたは 1 つのデータプロットの部分範囲にすることができます。

```
GraphLayer gl = Project.ActiveLayer();

DataPlot dp = gl.DataPlots(); // アクティブデータプロットを取得



DataRange dr;

int i1 = 0; // 最初のデータポイントから

int i2 = -1; // 最後のデータポイントまで

dp.GetDataRange(dr, i1, i2);
```

データ範囲オブジェクトでデータプロットから XY データを vector に取得します。

```
vector vx, vy;

DWORD dwRules = DRR_GET_DEPENDENT;

dr.GetData(dwRules, 0, NULL, NULL, &vy, &vx);
```

データ範囲を制御する

OriginC は、データ範囲を選択するために GetN ダイアログをサポートしています。

```
#include <GetNBox.h>

// ダイアログを開き、1 つのグラフデータプロットから範囲を選択

// この選択でデータ範囲オブジェクトを構築

GETN_TREE(tr)

GETN_INTERACTIVE(Range1, "Select Range", "")

if( GetNBox(tr) ) // OK ボタンで true を返す

{



        DataRange dr;

        dr.Add("Range1", tr.Range1.strVal);



        vector vData;

        int index = 0; // 範囲インデックス
```

```
        dr.GetData(&vData, index); // vData 内のデータは選択したデータポイント

}
```

9.2 文字列データ

9.2.1 文字列変数

```
string str1; // str1 という文字列変数を宣言

str1 = "New York"; // str1 に文字列シーケンスを割り当て



string str2 = "Tokyo"; // 文字列変数を宣言して割り当て



// 文字列配列を宣言して文字列シーケンスで初期化

char ch[] = "This is a test!";



// 文字配列を宣言し、サイズを指定して文字列シーケンスで初期化

char chArr[255] = "Big World.";
```

9.2.2 文字列を数値に変換

```
string str = PI; //  数値を文字変数に割り当て



// 文字列を数値に割り当て

double dd = atof(str, true);

out_double("dd=", dd); // dd=3.14159



// 文字列を複素数に変換

str = "1+2.5i";

complex cc = atoc(str);

out_complex("cc = ", cc); // cc = 1.000000+2.500000i



// 文字列を整数型に変換
```

```
str = "100";

int nn = atoi(str);

out_int("nn = ", nn); // nn = 100
```

9.2.3　数値/文字列を別の文字列に追加

String, Append StringString, Append Numeric

```
// 数値または文字列を別の文字列に追加

// Origin C では、,数値/文字列の定数または変数を加えるのに'+'を使う

string str = "The area is " + 30.7; // double 型の定数を文字列に追加



str += "¥n"; // 文字列定数を文字列変数に追加



int nLength = 10;

str += "The length is " + nLength; // 整数型変数を文字列に追加

out_str(str);
```

9.2.4　部分文字列を検索

String, FindString, Get Sub String

```
// 部分文字列を検索して取得

string str = "[Book1]Sheet1!A:C";

int begin = str.Find(']'); //  ']' のインデックスを検索して戻す

begin++; // ] の次の文字に移動



int end = str.Find('!', begin); // '!' のインデックスを検索して戻す

end--; // ! の次の文字に移動



// 開始インデックスと部分文字列の長さで部分文字列を取得

int nLength = end - begin + 1;

string strSheetName = str.Mid(begin, nLength);

out_str(strSheetName);// "Sheet1" が出力される
```

9.2.5　部分文字列の置換

String, Replace

```
// 1 文字を検索して置換

string str("A+B+C+");

int nCount = str.Replace('+','-');

out_int("", nCount); // nCount は 3

out_str(str); // "A-B-C-"



// 文字列を検索し、置換

str = "I am a student.¥nI am a girl.";

nCount = str.Replace("I am", "You are");

out_int("", nCount); // nCount は 2

out_str(str);
```

9.2.6　パス文字列関数

ファイルパスの文字列

```
// string::IsFile はファイルをチェックするのに使用

string strFile = "D:¥¥TestFolder¥¥abc.txt";

bool bb = strFile.IsFile();

printf("The file %s is %sexist.¥n", strFile, bb ?"" :"NOT ");



// GetFilePath 関数はフルパスの文字列からパスを抽出するのに使用

string strPath = GetFilePath(strFile);

out_str(strPath);



// GetFileName 関数はフルパスの文字列からファイル名の部分を

// 抽出するのに使用

bool bRemoveExtension = true;

string strFileName = GetFileName(strFile, bRemoveExtension);
```

```
out_str(strFileName);



// string::IsPath はパスが存在するかチェック

bb = strPath.IsPath();

out_int("", bb);
```

Origin システムパス

```
string strSysPath = GetOriginPath(ORIGIN_PATH_SYSTEM);

printf("Origin System Path:%s¥n", strSysPath);



string strUserPath = GetOriginPath(ORIGIN_PATH_USER);

printf("User File Path:%s¥n", strUserPath);
```

9.3　日付と時間データ

Origin C は、日付と時間データをサポートしています。

9.3.1　現在の日時データを取得

```
// 現在の時間を取得

time_t aclock;

time( &aclock );



// 時間の値を変換し、ローカル時間に修正

TM tmLocal;

convert_time_to_local(&aclock , &tmLocal);



// TM 形式からシステム時間形式に時間の値を変換

SYSTEMTIME sysTime;

tm_to_systemtime(&tmLocal, &sysTime);



// システム時間から日付文字列を取得
```

```
char       lpcstrTime[100];

if(systemtime_to_date_str(&sysTime, lpcstrTime, LDF_SHORT_AND_HHMM_SEPARCOLON))

    printf("Current Date Time is %s¥n", lpcstrTime);
```

9.3.2　ユリウス日を文字列に変換

```
SYSTEMTIME st;

GetSystemTime(&st); // 現在の日時を取得



double dJulianDate;

SystemTimeToJulianDate(&dJulianDate, &st); // ユリウス日に変換




// 指定した形式でユリウス日を文字列に変換

string strDate = get_date_str(dJulianDate, LDF_SHORT_AND_HHMM_SEPARCOLON);

out_str(strDate);
```

9.3.3　文字列をユリウス日に変換

```
string strDate = "090425 17:59:59";

double dt = str_to_date(strDate, LDF_YYMMDD_AND_HHMMSS);
```

10 プロジェクト

Origin C の Project クラスは、Origin プロジェクト内に含まれるさまざまな高いレベルのオブジェクトにアクセスするのに使用されます。これには、ワークブック、行列ブック、グラフ、ノート、フォルダなどが含まれます。Category:Project (class)

10.1 プロジェクト管理

Origin C は、プロジェクトを開く、保存する、追加する Project クラスを提供しており、プロジェクトに含まれるさまざまなオブジェクトにアクセスできます。Project クラスは、すべてのページタイプと一時データセットのコレクションを含みます。アクティブな曲線、レイヤ、フォルダなどのアクティブなオブジェクトを取得するメソッドがあります。

10.1.1 プロジェクトを開く/保存する

以下のコードは、プロジェクトを保存、新しいプロジェクトを開始、保存したプロジェクトを開くためのサンプルです。

```
string strPath = "c:¥¥abc.opj"; // プロジェクトのパスと名前

Project.Save(strPath); // 現在のプロジェクトを保存

Project.Open();        // 新しいプロジェクトを開始

Project.Open(strPath); //  保存したプロジェクトを開く
```

10.1.2 プロジェクトを別のプロジェクトに追加する

Project.Open メソッドの 2 番目の任意の引数を使って、プロジェクトを現在のプロジェクトに追加することができます。追加されたプロジェクトのフォルダ構造は、現在のプロジェクトのアクティブフォルダに置かれます。

```
Project.Open("c:¥¥abc.opj", OPJ_OPEN_APPEND);
```

10.1.3 修正フラグ

プロジェクトが編集されるとき、Origin によって内部的に **IsModified** フラグがセットされます。Origin C は、**IsModified** フラグをセットしたり、解除することができます。プロジェクトが閉じられると、このフラグにチェックが付きます。フラグがセットされると、Origin は変更を保存するかどうかをユーザに尋ねます。自分で作成した Origin C コードが保存する必要のない変更をしていたら、ユーザに尋ねる動作を行わないようにフラグを解除することができます。

```
if( Project.IsModified() )

{

    // アクティブプロジェクトを編集されていないようにセット次のことが分かっている場合これを行う

    // 変更を保存したくなく、Originに 保存されていない変更が

    // あることをユーザに通知したくない場合

    Project.ClearModified();
```

```
    // 新しいプロジェクトを開始、アクティブプロジェクトの保存されていない変更について

    // ユーザに尋ねないことがわかります。

    Project.Open();

}
```

10.2 フォルダを管理する

Origin プロジェクト内のページ(ワークブック、行列ブック、グラフ)は、階層型のフォルダ構造で管理でき、Origin のプロジェクトエクスプローラで表示されます。Origin C の Folder クラスは、フォルダを作成、アクティブ化、選択、配置することができます。

10.2.1 フォルダを作成し、そのパスを取得する

```
Folder fldRoot, fldSub;

fldRoot = Project.RootFolder;



// ルートフォルダにサブフォルダを名前を付けて追加

fldSub = fldRoot.AddSubfolder("MyFolder");

printf("Folder added successfully, path is %s¥n", fldSub.GetPath());
```

10.2.2 アクティブフォルダを取得する

```
Folder fldActive;

fldActive = Project.ActiveFolder();



// サブフォルダを追加

Folder fldSub;

fldSub = fldActive.AddSubfolder("MyFolder");

printf("Folder added successfully, path is %s¥n", fldSub.GetPath());
```

10.2.3 フォルダをアクティブにする

```
// ルートフォルダをアクティブ

Folder fldRoot = Project.RootFolder;

fldRoot.Activate();
```

```
// 指定したサブフォルダをアクティブ

Folder fldSub("/MyFolder");

fldSub.Activate();
```

10.2.4 指定したページのパスを取得

```
GraphPage gp("Graph1");

if( gp.IsValid() )

{

        Folder fld = gp.GetFolder();

        out_str(fld.GetPath());

}
```

10.2.5 ページ/フォルダを別の場所に移動する

Folder::Move を使って、ウィンドウ(ワークシート、グラフ...)やフォルダを別の場所に移動します。次のサンプルは、フォルダを移動する方法を示しています。

```
// 2 つのサブフォルダをルートフォルダに追加

Folder subfld1 = Project.RootFolder.AddSubfolder("sub1");

Folder subfld2 = Project.RootFolder.AddSubfolder("sub2");



// sub1 フォルダに sub2 フォルダを移動

if( !Project.RootFolder.Move(subfld2.GetName(), "/"+subfld1.GetName()+"/", true) )

    printf("move folder failed!");
```

10.3 ページにアクセスする

Origin のページはワークブック、行列ブック、グラフで構成され、プロジェクト内の中核的なオブジェクトです。Origin C は名前またはインデックスでページにアクセスでき、**foreach** ステートメントを使って現在のプロジェクトの特定のページタイプのすべてのインスタンスにアクセスできます。

10.3.1 名前とインデックスでページにアクセスする

すべてのページには名前があり、次のサンプルで示すように、ページにアクセスするために使用することができます。

```
// ページの名前でページにアクセス

GraphPage gp1("Graph1");
```

```
// 0 から始まるインデックスでページにアクセス

GraphPage gp2 = Project.GraphPages(0); // 最初のページは 0
```

10.3.2 アクティブページとレイヤを取得する

ワークブックページではレイヤはワークシート、グラフページではレイヤは対の軸、行列ページではレイヤは行列シートです。

アクティブレイヤのような、特定のレイヤに結びついているページにアクセスしたい場合、**Layer::GetPage** メソッドを使って行うことができます。

```
// アクティブレイヤを取得

GraphLayer gl = Project.ActiveLayer();



// レイヤからアクティブページを取得

GraphPage gp = gl.GetPage();
```

10.3.3 1 つのページをアクティベートする

ウィンドウをアクティベートしたいときは、PageBase::SetShow(PAGE_ACTIVATE) を使用します。

```
// Graph2 という名称のグラフウィンドウを付ける

GraphPage gp( "Graph2" );



// ウィンドウをアクティブに設定する

gp.SetShow( PAGE_ACTIVATE );
```

10.3.4 foreach を使う

foreach ステートメントは、コレクション内のすべての項目に対するループの処理を簡単にします。プロジェクトはさまざまなコレクションのすべてのページを含みます。

```
// 現在のプロジェクトのすべてのワークブックをループし

// 各ページの名前を出力

foreach( WorksheetPage wksPage in Project.WorksheetPages )

{

    out_str(wksPage.GetName());

}



// 現在のプロジェクトのすべての行列ブックをループし
```

```
// 各ページの名前を出力
foreach( MatrixPage matPage in Project.MatrixPages )
{
    out_str(matPage.GetName());
}

// 現在のプロジェクトのすべてのグラフをループし
// 各ページの名前を出力
foreach( GraphPage gp in Project.GraphPages )
{
    out_str(gp.GetName());
}

// 現在のプロジェクトのすべてのページをループし
// 各ページの名前を出力
foreach( PageBase pg in Project.Pages )
{
    out_str(pg.GetName());
}
```

10.4　メタデータにアクセスする

メタデータは、他のデータを参照する情報です。サンプルには、データが元々収集された時刻、データを収集した機器を操作した人、調べた標本の温度が含まれます。メタデータはプロジェクト、ページ、レイヤ、列に保存することができます。

10.4.1 データ範囲にアクセスする

Origin C の Project クラスは、Origin C の DataRange オブジェクトを現在のプロジェクトに追加、取得、削除するメソッドを提供しています。

```
Worksheet wks = Project.ActiveLayer();

DataRange dr;                    // 範囲オブジェクトを構成
dr.Add("X", wks, 0, 0, -1, -1); // ワークシート全体を範囲に追加
```

```
dr.SetName("Range1");            // 範囲名をセット

int UID = dr.GetUID(TRUE);       // 範囲オブジェクトの固有 ID を取得



int nn = Project.AddDataRange(dr); // 範囲をプロジェクトに追加
```

コマンドウィンドウまたはスクリプトウィンドウで、LabTalk コマンド **list r** を使って、現在のプロジェクトのすべての DataRange オブジェクトを一覧表示します。

10.4.2 アクセスツリー

プロジェクト内のツリーにアクセスする

ツリーの追加

このコードは、ツリー型の変数を宣言し、いくつかのデータをツリーのノードに割り当て、ツリーを現在のプロジェクトにツリーを追加します。

```
Tree tr;

tr.FileInfo.name.strVal = "Test.XML";

tr.FileInfo.size.nVal = 255;



// ツリー変数をプロジェクトに追加

int nNumTrees = Project.AddTree("Test", tr);

out_int("The number of trees in project:", nNumTrees);
```

ツリーを取得

同様に、似たようなコードは、Test という既存のツリー変数に保存され、*trTest* という新しいツリー変数に配置します。

```
// 名前でプロジェクトからツリーを取得

Tree trTest;

if( Project.GetTree("Test", trTest) )

        out_tree(trTest);
```

すべての LabTalk ツリーの名前を取得

Project::GetTreeNames メソッドは、プロジェクト内のすべての LabTalk ツリー変数の名前を取得します。ここで、名前は string vector に割り当てられ、割り当てられる文字列の数は、int 型で返されます。

```
vector<string> vsTreeNames;

int nNumTrees = Project.GetTreeNames(vsTreeNames);
```

ワークシート内のツリーにアクセスする

OriginObject::PutBinaryStorage が Origin オブジェクト、例えば、WorksheetPage, Worksheet, Column, GraphPage, MatrixPage などの多くの型にツリーを配置するのに使われます。

ツリーの追加

現在のプロジェクトにワークシートウィンドウを保持し、下記のサンプルコードを実行します。ユーザツリーを追加するコードを実行した後、ワークシートウィンドウのタイトルを右クリックし、オーガナイザの表示を選び、右側のパネルに追加したユーザツリーが表示されます。

```
Worksheet wks = Project.ActiveLayer();

if( wks )

{

        Tree tr;

        tr.name.strVal = "Jacky";

        tr.id.nVal = 7856;

        // wksTree というツリーをワークシートオブジェクトに配置

        string strStorageName = "wksTree";

        wks.PutBinaryStorage(strStorageName, tr);

}
```

ツリーを取得

OriginObject::GetBinaryStorage メソッドは、名前で Origin オブジェクトからツリーを取得するのに使われます。

```
Worksheet wks = Project.ActiveLayer();

if( wks )

{

        Tree tr;

        string strStorageName = "wksTree";

         // wksTree というツリーが存在していれば True を返す

        if( wks.GetBinaryStorage(strStorageName, tr) )

                out_tree(tr); // ツリーを出力

}
```

すべてのツリーの名前を取得

OriginObject::GetStorageNames メソッドは Origin オブジェクト内に保存されるすべての名前を取得します。2 つのストレージタイプ INI とバイナリがあります。ツリーはバイナリストレージに属し、以下のサンプルコードはワークシートからバイナリストレージを取得する方法を示します。

```
Worksheet wks = Project.ActiveLayer();

if( wks )

{

        // すべてのバイナリタイプのストレージの名前を取得

        vector<string> vsNames;

        wks.GetStorageNames(vsNames, STORAGE_TYPE_BINARY);

        for(int nn = 0; nn < vsNames.GetSize(); nn++)

                out_str(vsNames[nn]);

}
```

ワークシート列内のツリーにアクセスする

ワークシート列でツリーをセットおよび取得するには、上記で説明したようにワークシートでツリーをセットおよび取得するのと同じ方法を使います。

ツリーの追加

```
Worksheet wks = Project.ActiveLayer();

Column col(wks, 0);



Tree tr;

tr.test.strVal = "This is a column";

tr.value.dVal = 0.15;



col.PutBinaryStorage("colTree", tr);
```

ツリーを取得

```
Worksheet wks = Project.ActiveLayer();

Column col(wks, 0);



Tree tr;
```

```
if( col.GetBinaryStorage("colTree", tr) )

        out_tree(tr);
```

すべてのツリーの名前を取得

```
Worksheet wks = Project.ActiveLayer();

Column col(wks, 0);



// すべてのバイナリタイプのストレージの名前を取得

vector<string> vsNames;

col.GetStorageNames(vsNames, STORAGE_TYPE_BINARY);



for(int nn = 0; nn < vsNames.GetSize(); nn++)

        out_str(vsNames[nn]);
```

ファイルインポートのツリーノードにアクセスする

ワークシートにデータをインポートしたあと、Origin はページレベルで特別なツリーのような構造のメタデータを保存します。ファイルについての基本情報は、ツリーから直接取り出したり、配置することができます。

```
Worksheet wks = Project.ActiveLayer();

WorksheetPage wksPage = wks.GetPage();



storage st;

st = wksPage.GetStorage("system");



Tree    tr;

tr = st;



double dDate = tr.Import.FileDate.dVal;

printf("File Date:%s¥n", get_date_str(dDate, LDF_SHORT_AND_HHMMSS_SEPARCOLON));

printf("File Name:%s¥n", tr.Import.FileName.strVal);

printf("File Path:%s¥n", tr.Import.FilePath.strVal);
```

レポートシートツリーにアクセスする

分析レポートシートは、ツリー構造に基づく特別な形式のワークシートです。次のように、レポートシートからレポートツリーを取得できます。

```
Worksheet wks = Project.ActiveLayer();



Tree trReport;

uint uid; // レポート範囲の UID を受け取る

// エスケープ操作文字列を変換する(ex.?$OP:A=1)

// 戻りのツリーに実際のデータセット名



bool bTranslate = true;

if( wks.GetReportTree(trReport, &uid, 0, GRT_TYPE_RESULTS, true) )

{

        out_tree(trReport);

}
```

10.5 操作にアクセスする

10.5.1 すべての操作のリスト

列の統計や非線形曲線フィットなど、多くのダイアログにある再計算分析ツールは、Operation クラスに基づいています。操作全体が完了したら、結果シートまたは結果グラフに錠前アイコンが現れます。すべての操作を **Project::Operations** を使って、一覧表示することができます。次のコードは、すべての操作を取得し、操作の名前を印刷するのに使うことができます。

```
OperationManager opManager;

opManager = Project.Operations;

int count = opManager.GetCount();

for(int index=0; index < count; index++)

{

        OperationBase& op = opManager.GetOperation(index);

        string strName = op.GetName();

        out_str(strName);

}
```

10.5.2 ワークシートが階層であるかチェック

ワークシートが結果テーブルのシートであるかどうかをチェックする場合、次のコードのようにレイヤシステムパラメータを使ってチェックすることができます。

```
Worksheet wks = Project.ActiveLayer();

bool bHierarchySheet = (wks.GetSystemParam(GLI_PCD_BITS) & WP_SHEET_HIERARCHY);

if( bHierarchySheet )

        out_str("This is a report table sheet");

else

        out_str("This is not a report table sheet");
```

10.5.3 レポートシートにアクセスする

次のコードは、レポートシートからレポートツリーを取得し、レポートツリーから得られる結果を文字フォーマットにリンクしたセルに変換し、新しいワークシートに配置する方法を示しています。

これはレポートシートからレポートツリーを取得する方法です。このコードを実行するには、レポートシートをアクティブにする必要があります。

```
Worksheet wks = Project.ActiveLayer();

Tree trResult;

wks.GetReportTree(trResult);
```

次のコードは、レポートツリーから必要な結果を取得し、それらを文字フォーマットにリンクしたセルに変換し、新しく作成したワークシートに配置する方法を示しています。

```
// サマリー表に新しいシートを追加

WorksheetPage wksPage = wks.GetPage();

int index = wksPage.AddLayer();

Worksheet wksSummary = wksPage.Layers(index);



string strCellPrefix;

strCellPrefix.Format("cell://%s!", wks.GetName());



vector<string> vsLabels, vsValues;

// パラメータ

vsLabels.Add(strCellPrefix + "Parameters.Intercept.row_label2");
```

```
vsValues.Add(strCellPrefix + "Parameters.Intercept.Value");
vsLabels.Add(strCellPrefix + "Parameters.Slope.row_label2");
vsValues.Add(strCellPrefix + "Parameters.Slope.Value");

// 統計
vsLabels.Add(strCellPrefix + "RegStats.DOF.row_label");
vsValues.Add(strCellPrefix + "RegStats.C1.DOF");
vsLabels.Add(strCellPrefix + "RegStats.SSR.row_label");
vsValues.Add(strCellPrefix + "RegStats.C1.SSR");

// 列に配置
Column colLabel(wksSummary, 0);
Column colValue(wksSummary, 1);
colLabel.PutStringArray(vsLabels);
colValue.PutStringArray(vsValues);
```

11 インポート

Origin の大きな利点の 1 つは、異なる形式のデータをワークシートや行列シートにインポートできることです。Origin C は、ASCII、バイナリファイル、画像ファイル、データベース内のデータをインポートする機能があります。次のセクションでは、ワークシートまたは行列シートにデータをインポートする方法を示しています。

11.1 データをインポートする

Worksheet および MatrixLayer クラスは Datasheet クラスから派生されます。Datasheet クラスには ImportASCII というメソッドがあります。importASCII メソッドは、ASCII データファイルをインポートするのに使われます。Microsoft Excel および SPC データファイルをそれぞれインポートするには、ImportExcel および ImportSPC というメソッドがあります。

Import ASCII Data File to Worksheet

11.1.1 ASCII データファイルをワークシートにインポート

最初のサンプルは、ASCII データファイルをアクティブワークブックのアクティブワークシートにインポートします。最初にファイルフォーマットを検出するために、AscImpReadFileStruct グローバル関数を呼び出します。フォーマット情報は、ASCIMP 構造に保存されています。そして構造は、実際にインポートを行う ImportASCII メソッドに渡されます。Import ASCII Data File to Worksheet

```
string strFile = "D:¥¥data.dat"; // いくつかのデータファイル名

ASCIMP  ai;

if(0 == AscImpReadFileStruct(strFile, &ai) )

{

    // このサンプルでは、LabTalk のシステム変数@NPO を 0 にして

    // ASCII インポートのプログレスバーを無効

    // これは任意で、ここで可能であることを表示

    // LTVarTempChange クラスは

    // LabTalk 変数を簡単に設定および元に戻す LTVarTempChange についての詳細は

    // LabTalk にアクセスするセクションをご覧ください

    LTVarTempChange progressBar("@NPO", 0); // 0 = プログレスバーを無効化

    // アクティブワークブックからアクティブワークシートを取得

    Worksheet wks = Project.ActiveLayer();

    if(0 == wks.ImportASCII(strFile, ai))

        out_str("Import data successful.");

}
```

次のサンプルは、ASCII データファイルをワークシートにインポートしますが、ファイルから各列の情報も取得し、ワークシート列をセットアップします。

```
// ファイルを開くダイアログでユーザにインポートするファイルの選択を促す

string  strFile = GetOpenBox("*.dat");

if( strFile.IsEmpty() )

    return; // ユーザがキャンセルあるいはエラー



ASCIMP  ai;

if( 0 == AscImpReadFileStruct(strFile, &ai) )

{

    ai.iAutoSubHeaderLines = 0; // 自動検出サブヘッダを無効化



    // 1. ロングネーム

    // 2.単位

    // 3.拡張された説明（ユーザ定義）

    // 4.データ型の指示（ユーザ定義）

    ai.iSubHeaderLines = 4;



    // iAutoSubHeaderLines が false(0)のとき、ai.nLongName, ai.nUnits, ai.nFirstUserParams の

    // 開始インデックスがメインヘッダから取得

    ai.nLongNames = ai.iHeaderLines;

    ai.nUnits = ai.iHeaderLines + 1;



    // 最初のユーザパラメータのインデックスをセット

    ai.nFirstUserParams = ai.iHeaderLines + 2;

    ai.nNumUserParams = 2; // ユーザパラメータの数をセット



    // コメントラベルにヘッダをセットしない
```

```
    ai.iMaxLabels = 0;



    // アクティブワークブックからアクティブワークシートを取得

    Worksheet wks = Project.ActiveLayer();



    if( 0 == wks.ImportASCII(strFile, ai) ) // エラーなしは 0 を返す

    {

        // ユーザパラメータラベルの名前

        vector<string> vsUserLabels = {"Expanded Description", "Type Indication"};



        // ユーザパラメータラベルを指定した名前にセット

        Grid grid;

        grid.Attach(wks);

        grid.SetUserDefinedLabelNames(vsUserLabels);

        wks.AutoSize(); // 内容に合うように列幅を調整

    }

}
```

11.1.2 ASCII データファイルを行列にインポート

データを行列シートにインポートすることは、ワークシートへのインポートに似ています。この例は、最初のワークシート例に似ています。違いは、Worksheet クラスではなく、MatrixLayer クラスを使って、アクティブな行列ブックからアクティブな行列シートを取得するという点です。

```
string strFile = "D:¥¥someData.dat";

ASCIMP ai;

if( 0 == AscImpReadFileStruct(strFile, &ai) )

{

    MatrixLayer ml = Project.ActiveLayer();

    if( 0 == ml.ImportASCII(strFile, ai) )

        out_str("Data imported successfully.");

}
```

11.1.3 インポートフィルタを使ってデータをインポート

ファイルインポートのための関数は、**OriginC\Originlab\FileImport.h** ファイル内で宣言されます。これらの関数は、Origin C の言語リファレンスヘルプに説明があります。

ファイルインポート関数を呼ぶ前に、最初にプログラムで **FileImport.c** をロードし、コンパイルする必要がありますこれは、コマンドを使ってスクリプトから行うことができます。

```
run.LoadOC(Originlab¥FileImport.c, 16);

// 16 というオプションは、FileImport.c ファイルの corresponding .h で

// スキャンすることで、すべての Origin C 依存ファイルがロードされるようにする
```

次のサンプルは、フィルタファイルでデータをインポートすることを示しています。

```
#include <..¥Originlab¥FileImport.h>

void import_with_filter_file()

{

    Page pg = Project.Pages(); // アクティブページ



    // ページブック名を取得

    string strPageName = pg.GetName();



    // ページのアクティブインデックスを取得

    int nIndexLayer = pg.Layers().GetIndex();



    // Origin のサンプルフォルダを取得

    string strPath = GetAppPath(TRUE) + "Samples¥¥Signal Processing¥¥";



    // .oif フィルタ名を指定

    string strFilterName = "TR Data Files";



    import_file(strPageName, nIndexLayer, strPath + "TR2MM.dat", strFilterName);

}
```

データフォーマットに合わせるために、既存のフィルタを修正し、ファイルからフィルタをロードし、設定する必要があるかもしれません。次のケースを確認してください。

```
#include <..¥Originlab¥FileImport.h>

void    config_filter_tree()

{

    string strFile = GetAppPath(1) + "Samples¥¥Curve Fitting¥¥Step01.dat";

    if( !strFile.IsFile() )

        return;



    // ツリーにフィルタをロード

    Tree trFilter;

    string strFilterName = "ASCII";

    int nLocation = 1; // 組み込みのフィルタフォルダ

    Worksheet wks;

    wks.Create("origin");

    WorksheetPage wp = wks.GetPage();

    string strPageName = wp.GetName();

    int nRet = load_import_filter(strFilterName, strFile,

                                  strPageName, nLocation, trFilter);

    if( 0 != nRet )

        out_str("Failed to load import filter");



    // フィルタツリー更新

    trFilter.iRenameCols.nVal = 0; // 0 はデフォルトの列名を保持、1 は列の名前を変更



    // フィルタツリーでデータファイルをインポート

    // import_files 関数は、一度に複数ファイルのインポートをサポート

    vector<string> vsDataFileName;

    vsDataFileName.Add(strFile);

    nRet = import_files(vsDataFileName, strPageName, wks.GetIndex(), trFilter);
```

```
    if( 0 != nRet )

        out_str("Failed to import file");

}
```

11.1.4 インポートウィザードでファイルをインポート

インポートするファイルが、ASCII ファイルでも単純なバイナリファイルでもない場合、あるいは、データファイルのインポートフィルタがない場合は、Origin C の impFile X ファンクションを使用し、インポートウィザードウィザードでファイルをインポートできます。

Origin C 関数は、次のプロトタイプのどちらかを使います。

int YourFunctionName(Page& pgTarget, TreeNode& trFilter, LPCSTR lpcszFile, int nFile)

ここで:

- **pgTarget:**worksheet 型または Matrix 型の Page オブジェクトへの参照これは、自分のフィルタ内またはインポートウィザードの「データソース」ページでターゲットウィンドウとして定義するものです。

- **trFilter:**フィルタファイルまたはウィザードの指定からのすべてのフィルタ設定を持つツリー構造での TreeNode オブジェクトへの参照。

- **lpcszFile:**インポートするファイルのフルパスと名前

- **nFile:**インポートするファイルの順序を示すファイルインデックス番号 (例えば、n 個のファイルをインポートする場合、関数は n 回呼び出され、nFile は処理されるファイルの数となります。

または

int YourFunctionName(Layer& lyTarget, TreeNode& trFilter, LPCSTR lpcszFile, int nFile)

ここで:

- **lyTarget:**worksheet 型または Matrix 型の Layer オブジェクトへの参照。これは、自分のフィルタ内またはインポートウィザードの「データソース」ページでターゲットウィンドウとして定義するものです。

- **trFilter:**フィルタファイルまたはウィザードの指定からのすべてのフィルタ設定を持つツリー構造での TreeNode オブジェクトへの参照。

- **lpcszFile:**インポートするファイルのフルパスと名前

- **nFile:**インポートするファイルの順序を示すファイルインデックス番号 (例えば、n 個のファイルをインポートする場合、関数は n 回呼び出され、nFile は処理されるファイルの数となります。

Origin のインストールフォルダの、\Samples\Import と Export\User Defined フォルダにサンプルがあります。

Note:インポートウィザードの最初のページ(「**データソース**」ページ)にあるターゲットウィンドウテンプレートの名前は、新しいウィンドウを作成するときだけ使われます(ドラッグ&ドロップインポート時にはいくつか条件があります。)。**ファイル**:**インポート**を選択するとき、アクティブウィンドウがインポートフィルタの**ターゲットウィンドウ**として設定されていると、新しいウィンドウは作成されず、アクティブウィンドウへの page オブジェクトの参照が関数に渡されます。アクティブウィンドウが異なるタイプの場合、指定したテンプレートで新しいウィンドウが作成され、この新しいウィンドウへの page 参照が関数に渡されます。

インポートウィザードで変数を拡張

インポートウィザードで ASCII ファイルをインポートするとき、ユーザ定義の Origin C 関数を使ってファイルヘッダから変数を抽出することができます。

Origin C 関数は、次のようなプロトタイプになっている必要があります。

int FuncName(StringArray& saVarNames, StringArray& saVarValues, const StringArray& saHdrLines, const TreeNode &trFilter);

ここで:

- **saVarNames**: 変数名を配置する文字列配列
- **saVarValues**:変数値を配置する文字列配列
- **saHdrLines**:ヘッダ行を含む文字列配列への参照。ヘッダ行が自動的に関数に渡されるので、Origin C 関数はデータファイルを読み込む必要はありません。
- **trFilter**:フィルタファイルまたはウィザードの指定からのすべてのフィルタ設定を持つツリー構造での TreeNode オブジェクトへの参照。

11.2 画像のインポート

Origin は、行列、ワークシートセル、グラフにイメージをインポートすることができます。次のセクションは、Origin C アプリケーションでイメージをインポートする方法を示します。

11.2.1 イメージを行列にインポート

次のサンプル関数は、イメージファイルを行列にインポートする方法を示しています。関数は、行列名、ファイル名、グレースケールの深度の 3 つの引数を取ります。このサンプルで呼び出されるキーとなる関数は、oimg_image_info および oimg_load_image です。前者はイメージファイルに含まれるイメージについての情報を取得するのに使われます。取得した情報は、目的の行列を準備するのに使われます。後者の関数は、グレースケールデータ値として目的の行列にイメージファイルに実際にインポートするのに使われます。

```
#include <import_image.h> // oimg_ functions で必要



bool import_image_to_matrix_data(
```

```
    LPCSTR lpcszMatrixName, // 行列ブック名
    LPCSTR lpcszFileName,   // 画像ファイル名
    int nGrayDepth)         // 8 ビットまたは 16 ビットのグレースケールとしてインポート
{
    // 目的の行列オブジェクトを取得
    MatrixObject mo(lpcszMatrixName);
    if( !mo.IsValid() )
        return false;

    // 元のイメージ情報を取得
    int nWidth, nHeight, nBPP;
    if( !oimg_image_info(lpcszFileName, &nWidth, &nHeight, &nBPP) )
        return false;

    // 目的の行列を元の画像を同じ次数にセット
    if( !mo.SetSize(nHeight, nWidth, 0) )
        return false;

    // 目的の行列のデータサイズ
    int nDataType = (16 == nGrayDepth ?FSI_USHORT :FSI_BYTE);
    if( !mo.SetInternalData(nDataType, FALSE, FALSE) )
        return false;

    // イメージを行列にインポート
    bool bRet;
    if( FSI_USHORT == nDataType )
    {
        Matrix<WORD>& mm = mo.GetDataObject();
```

```
        bRet = oimg_load_image(lpcszFileName, &mm, 16, nHeight, nWidth);
    }
    else // FSI_BYTE
    {
        Matrix<BYTE>& mm = mo.GetDataObject();
        bRet = oimg_load_image(lpcszFileName, &mm, 8, nHeight, nWidth);
    }
    return bRet;
}
```

11.2.2 イメージをワークシートに挿入する

次のサンプルは、ファイルから JPEG ファイルをワークシートセルに埋め込みます。これは、Worksheet クラスの AttachPicture メソッドを使って行います。

```
int nRow = 0, nCol = 0;
string strFile = "D:¥¥Graph1.jpg";
DWORD dwEmbedInfo = EMBEDGRAPH_KEEP_ASPECT_RATIO;

Worksheet wks = Project.ActiveLayer();
if( wks.AttachPicture(nRow, nCol, strFile, dwEmbedInfo) )
{
    wks.Columns(nCol).SetWidth(20);
    wks.AutoSize();
}
```

11.2.3 イメージをグラフにインポートする

次のサンプルは、ファイルから JPEG ファイルをグラフレイヤに埋め込みます。これは、**image_import_to_active_graph_layer** グローバル関数を使って行われます。

```
#include <image_utils.h>

// image_import_to_active_graph_layer 関数を呼び出す前に
// image_utils.c をコンパイル
```

```
LT_execute("run.LoadOC(Originlab¥¥image_utils.c)");



string strFile = "D:¥¥Graph1.jpg";

image_import_to_active_graph_layer(strFile);
```

11.3 動画のインポート

11.3.1 バージョン情報

必要な Origin のバージョン:Pro 93 SR0

Origin C は、ビデオファイルの読み取りと動画のフレームの行列へのインポートのために VideoReader クラスを提供しています。このクラスは OriginPro でのみ利用できます。

この VideoReader クラスを使用するために、ヘッダファイル"VideoReader.h" をソースコードにインクルードする必要があります。

```
#include <..¥OriginLab¥VideoReader.h>
```

VideoReader クラスで、動画ファイルを開き、フレーム数、フレーム率(フレーム/秒)、現在の位置などのプロパティを取得できます。また、検索フレーム、検索時間、行列オブジェクトへの読み込みフレームのためのメソッドを提供しています。

次のサンプルでは、新しい行列ブックを作成し、動画内の 10 フレームを探します。そして、他のフレームをスキップすることにより、100 フレームを行列オブジェクトとして、アクティブ行列シートにロードします。

```
#include <..¥Originlab¥VideoReader.h>  // ヘッダファイルをインクルード

void Import_Video_Ex1(string strFile = "d:¥test.avi") {

        MatrixLayer ml;

        ml.Create("Origin");  // フレームのための行列シートを作成

        char str[MAXLINE];

        VideoReader vr;  // VideoReader を宣言

        strcpy(str, strFile);

        if(!vr.Open(str)) {  // 動画ファイルを開く

                out_str("Failed to open video file!");

                return;

        }

        // フレーム数取得

        int iFrameCount = (int)vr.GetFrameCount();

        printf("%u frames¥n",iFrameCount);
```

```
        // 開始フレーム
        int iOffset = 10;
        // 読み込むフレームの合計を指定
        int iTotalFrames = 100;
        // 各読み込み間でスキップするフレームを指定
        int iSkip = 1; // 全ての他のフレームを読み込む
        bool bRet = vr.SeekFrame(iOffset);
        vr.ReadFrames(ml, iTotalFrames, iSkip);  // フレームを読み込む
        if(vr.ReaderReady()) {
                vr.Close();  // ビデオリーダを閉じる
        }
}
```

このサンプルでは、時間が測定基準として使用されています。

```
#include <..¥Originlab¥VideoReader.h>  // ヘッダファイルをインクルード
void Import_Video_Ex2(string strFile = "d:¥test.avi") {
        MatrixLayer ml;
        ml.Create("Origin");  // フレームのための行列シートを作成
        char str[MAXLINE];
        VideoReader vr;  // VideoReader を宣言
        strcpy(str, strFile);
        if(!vr.Open(str)) {  // 動画ファイルを開く
                out_str("Failed to open video file!");
                return;
        }
        // フレーム数を取得
        int iFrameCount = (int)vr.GetFrameCount();
        // フレーム率を取得
        double dFPS = vr.GetFPS();
```

```
        double dRunningTime = iFrameCount / dFPS;
        printf("%u frames at %f fps with a running time of %f seconds¥n",
                iFrameCount, dFPS, dRunningTime);

        // 読み込みのためのセットアップ
        double dStartTime = 5; // 5 秒で読み込み開始
        double dSkipLength = 3.333; // 読み込みの間で 3.333 秒スキップ
        vr.SeekFrame((int) dStartTime * dFPS); // フレーム開始を計算
        int iSkip = (int) dSkipLength * dFPS; // スキップするフレーム数を計算
        // 実際に読み込むフレームの数を計算
        int iTotalFrames = (int) ( (dRunningTime - dStartTime) * dFPS)
        / (iSkip + 1);
        vr.ReadFrames(ml, iTotalFrames, iSkip);  // フレームを読み込み
        if(vr.ReaderReady()) {
                vr.Close();  // ビデオリーダを閉じるｓ
        }
}
```

12 エクスポート

12.1 ワークシートのエクスポート

Worksheet クラスは、ワークシートデータをファイルに保存する メソッドがあります。メソッドは、開始行と開始列、終了行と終了列を指定する引数を持ちます。また、欠損データ値を取り扱う方法や列ラベルをエクスポートするかどうかを指定することもできます。OriginC:Worksheet-ExportASCII

以下のサンプルはすべて wks が有効な **Worksheet** オブジェクトで、strFileName が目的のファイルへのフルパスおよびファイル名を含む **string** オブジェクトであるものとします。

最初のサンプルは、ワークシート内のデータをすべて TAB 区切りとして、欠損値をスペースとしてファイルに保存するものです。

```
wks.ExportASCII(strFileName,

    WKS_EXPORT_ALL|WKS_EXPORT_MISSING_AS_BLANK);
```

次のサンプルは、ワークシート内のデータをすべてカンマ区切りとして、欠損値をスペースとしてファイルに保存するものです。さらに列ラベルも保存されます。

```
wks.ExportASCII(strFileName,

    WKS_EXPORT_ALL|WKS_EXPORT_LABELS|WKS_EXPORT_MISSING_AS_BLANK,

    ',');
```

最後のサンプルは、ワークシート内のデータの最初の 2 列をカンマ区切りとして、欠損値をスペースとしてファイルに保存するものです。さらに列ラベルも保存されます。行と列のインデックスは 0 から始まります。終了行と終了列のインデックスは、それぞれ最終行と最終列を示すため -1 にすることができます。

```
wks.ExportASCII(strFileName,

    WKS_EXPORT_ALL|WKS_EXPORT_LABELS|WKS_EXPORT_MISSING_AS_BLANK,

    '¥t',

    0, 0,    // 最初の行、最初の列から開始

    -1, 1); // 最終行、2 列目で終了
```

12.2 グラフのエクスポート

Origin は、グラフをさまざまな画像形式にエクスポートすることができます。Origin C は、**export_page** および **export_page_to_image** 関数でこの機能にアクセスできます。

次のサンプルは、プロジェクト内のすべてのグラフを EMF ファイルにエクスポートします。EMF ファイルの名前は、グラフの名前と同じで、C ドライブのルートに出力されます。

```
string strFileName;

foreach(GraphPage gp in Project.GraphPages)
```

```
{

    strFileName.Format("c:¥¥%s.emf", gp.GetName());

    export_page(gp, strFileName, "EMF");

}
```

次のサンプルは、800x600 の JPEG ファイルにアクティブグラフをエクスポートします。JPEG ファイルの名前は、グラフの名前と同じで、C ドライブのルートに出力されます。

```
GraphPage gp;

gp = Project.ActiveLayer().GetPage();

if( gp ) // アクティブページがグラフの場合

{

    string strFileName;

    strFileName.Format("c:¥¥%s.emf", gp.GetName());

    export_page_to_image(strFileName, "JPG", gp, 800, 600, 8);

}
```

12.3 行列のエクスポート

Origin の Matrix は、ASCII データファイルまたは画像ファイルにエクスポートできます。

12.3.1 行列を ASCII データファイルにエクスポート

次のサンプルは、アクティブな行列ウィンドウから*.txt ファイルに ASCII データをエクスポートする方法を示しています。**export_matrix_ascii_data** 関数に対し、*#include <oExtFile.h>* を追加する必要があります。

```
file    ff;

if ( !ff.Open("C:¥¥ExpMatData.txt", file::modeCreate|file::modeWrite) )

    return; //書き込みのためのファイルを開くのを失敗



string          strRange;

MatrixLayer ml = Project.ActiveLayer();

ml.GetRangeString(strRange);



LPCSTR                  lpcszSep = "¥t";

vector<string>  vXLabels, vYLabels; // 空はラベルなしの意味
```

```
DWORD                   dwCntrl = GDAT_FULL_PRECISION | GDAT_MISSING_AS_DASHDASH;



// エラーなしで 0 を返す

int nErr = export_matrix_ascii_data(&ff, strRange, ml.GetNumRows(),

        ml.GetNumCols(), lpcszSep, &vXLabels, &vYLabels, dwCntrl);
```

12.3.2 行列からのイメージを画像ファイルにエクスポート

次のサンプルは、行列を画像ファイルにエクスポートする方法を示します。

サンプルを実行する前に image_utils.c ファイルをロードしてコンパイルしておく必要があります。これは、次のコマンドを実行するか、ワークスペース内にこのファイルを追加して行います。

```
run.LoadOC(Originlab¥image_utils.c);
```

export_Matrix_to_image 関数のために、*#include <image_utils.h>* を追加する必要があります。

```
MatrixLayer ml = Project.ActiveLayer();

MatrixObject mo = ml.MatrixObjects();

export_Matrix_to_image("c:¥¥matrixImg.jpg", "jpg", mo);
```

12.4 動画のエクスポート Video Writer

12.4.1 バージョン情報

必要な Origin のバージョン:Origin 9 SR0

Origin では、グラフのコレクションを動画として作成することが可能です。Origin C では、ビデオライターを使用した機能にアクセスし、圧縮のためのコーデックを定義(詳細は FourCC 表を参照)して、動画の名前やパス、スピードと寸法、フレームとしてのグラフページを指定して作成できます。

Note:ビデオライターを使用するには、ヘッダファイルをインクルードします。

```
#include <..¥OriginLab¥VideoWriter.h>
```

次のサンプルは、プロジェクト内の各グラフをフレームとして動画に書き込み、2 フレーム/秒のスピード、800px*600px のサイズの動画を作成します。

```
// 圧縮なしで元のフォーマットを使用

    int codec = CV_FOURCC(0,0,0,0);



    // VideoWriter object オブジェクトを作成

    VideoWriter vw;

    int err = vw.Create("D:¥¥example.avi", codec, 2, 800, 600);
```

```
if (0 == err)
{
  foreach(GraphPage grPg in Project.GraphPages)

  // ビデオにグラフページを書き込み
  err = vw.WriteFrame(grPg);
}

// 完了したらビデオオブジェクトをリリース
vw.Release();

return err;
```

次のサンプルでは、グラフページを動画に個別に書き込み、動画内のこのグラフページのフレーム数を定義します。

```
GraphPage gp("Graph1");

// 定義されたグラフページは 10 フレーム続く
int nNumFrames = 10;

vw.WriteFrame(gp, nNumFrames);
```

13 分析とアプリケーション

Origin C は、データ分析、数学や科学技術の業務で利用できる価値ある機能をサポートしています。次のセクションでは、これらの関数について、利用分野に応じたサンプルを提供しています。

13.1 数学

13.1.1 規格化

次のサンプルは、データプロット(曲線)内のデータポイントを選択し、レイヤ内のすべての曲線をその点と同じ値に正規化します。このコードは、複数曲線を持つグラフレイヤがアクティブであり、すべての曲線が同じ X 値を共有していることを前提としています。この前提条件は、スペクトル分析ではよく行われることです。

```
GraphLayer gl = Project.ActiveLayer();

if( !gl )

    return;



// ある特定の曲線の 1 つの特定のポイントをクリックして選択

GetGraphPoints mypts;

mypts.SetFollowData(true);

mypts.GetPoints(1, gl);



vector vx, vy;

vector<int> vn;

if(mypts.GetData(vx, vy, vn) == 1)

{

    // 選択したポイントのインデックスと y 値を保存

    int nxpicked = vn[0] - 1;

    double dypicked = vy[0];



    // レイヤ内のすべてのデータポイントをループ

    foreach( DataPlot dp in gl.DataPlots )
```

```
    {
        // データ範囲、現在のプロットの y 列
        XYRange xy;
        Column cy;
        if(dp.GetDataRange(xy) && xy.GetYColumn(cy))
        {
            // y 列から vector 参照を y 値に取得
            vectorbase &vycurrent = cy.GetDataObject();

            // vector をスケールし、y 値と選択したポイントを一致
            vycurrent *= dypicked/vycurrent[nxpicked];
        }
    }
}
```

13.1.2 補間/補外

ocmath_interpolate 関数は、線形、スプライン、B スプラインのモードで補間/補外を実行するのに使われます。

```
// アクティブワークシートを 4 列にする
// 最初の 2 列に元の xy データ
// 3 列目に入力 x 値、4 列目に y 値
Worksheet    wks = Project.ActiveLayer();
wks.SetSize(-1, 4);

DataRange drSource;
drSource.Add(wks, 0, "X"); // 1 列目 - 元の x データ
drSource.Add(wks, 1, "Y"); // 2 列目 - 元の y データ

vector vSrcx, vSrcy;
drSource.GetData(&vSrcx, 0);
```

```
drSource.GetData(&vSrcy, 1);



DataRange drOut;

drOut.Add(wks, 2, "X"); // 3 列目 - 入力 x データ

drOut.Add(wks, 3, "Y"); // 4 列目 - 補間した y データ



vector vOutx, vOuty;

drOut.GetData(&vOutx, 0);



int    nSrcSize = vSrcx.GetSize();

int    nOutSize = vOutx.GetSize();

vOuty.SetSize(nOutSize);



int nMode = INTERP_TYPE_BSPLINE;

double dSmoothingFactor = 1;

int iRet = ocmath_interpolate(vOutx, vOuty, nOutSize, vSrcx, vSrcy, nSrcSize,

nMode, dSmoothingFactor);



drOut.SetData(&vOuty, &vOutx);
```

13.1.3 積分

Origin C では、積分を実行するのに NAG の積分ルーチンを使っています。Origin C と NAG を使って、被積分関数、パラメータを持つ被積分関数、揺れのある被積分関数、不定積分、高次積分などで積分を実行できます。次のサンプルは、NAG を使った積分を行う方法を示しています。

Origin C コードには、NAG 関数を呼ぶ前に NAG ヘッダファイルをインクルードします。

```
#include <OC_nag.h> // NAG の宣言
```

簡単な積分関数

最初のサンプルは、1 つの積分変数を持つ簡単な被積分関数で基本積分を実行する方法を示しています。

```
// NAG_CALL は適切な呼び出し表記これを関数ポインタのように扱い

// 自分自身の被積分関数を定義
```

```
double NAG_CALL func(double x)
{
    return (x*sin(x*30.0)/sqrt(1.0-x*x/(PI*PI*4.0)));
}

void nag_d01ajc_ex()
{
        double a = 0.0;
        double b = PI * 2.0;  // 積分間隔

        double epsabs, abserr, epsrel, result;
        // 精度が十分でない場合、epsabs と epsrel を使って
        // この量を目的の精度に拡張
        epsabs = 0.0;
        epsrel = 0.0001;

        // 積分内の関数を評価するために
        // 部分間隔の最大数が必要。ほとんどの場合、200 から 500 くらいが適切であり、お薦め。
        int max_num_subint = 200;

        Nag_QuadProgress qp;
        NagError fail;

        d01ajc(func, a, b, epsabs, epsrel, max_num_subint, &result, &abserr,
        &qp, &fail);

        // 次の 3 つのエラー以外のエラーは
```

```
        // 入力が不正か、割り当ての失敗再び積分ルーチンを呼び出す前に
        // メモリリークを避け、メモリ割り当てを
        // 解放する必要がある
        if (fail.code != NE_INT_ARG_LT && fail.code != NE_BAD_PARAM &&
            fail.code !=  NE_ALLOC_FAIL)
        {
                NAG_FREE(qp.sub_int_beg_pts);
                NAG_FREE(qp.sub_int_end_pts);
                NAG_FREE(qp.sub_int_result);
                NAG_FREE(qp.sub_int_error);
        }

        printf("%g¥n", result);
}
```

パラメータ付きの積分関数

次のサンプルは、パラメータ付きの被積分関数で積分を定義し、実行する方法を示しています。ユーザ定義の構造でパラメータが積分子に渡されます。これは、スタティックな変数を被積分関数のパラメータとして使用し、スレッドを安全にします。

このサンプルは、NAG の不定積分子を使用することもできます。例えば、不定積分 **d01smc** 関数を呼び出す行を有効にして、サンプルが不定積分を実行するのに使用します。

```
struct user //被積分パラメータ
{
    double A;
    double Xc;
    double W;
};

// ユーザによって供給される関数は、与えられた x で被積分関数の戻り値を返す

static double NAG_CALL f_callback(double x, Nag_User *comm)
```

```
{
    struct user *param = (struct user *)(comm->p);

    return param->A * exp(-2 * (x - param->Xc) * (x - param->Xc)
        / param->W / param->W) / (param->W * sqrt(PI / 2));
}
```

関数に対するパラメータをセットし、積分を実行する必要のある追加のパラメータを定義します。そして、積分は、引数としてパラメータを渡し、1 つの関数呼び出しで実行されます。

```
void nag_d01sjc_ex()
{
    double a = 0.0;
    double b = 2.0; // 積分間隔

    // 次の変数は積分の精度を
    // 制御するのに使用
    double epsabs = 0.0;      // 絶対精度、相対精度を使うには負値をセット
    double epsrel = 0.0001;   // 相対精度、絶対精度を使うには負値をセット
    int max_num_subint = 200; //  最大部分間隔をセット、200-500 がお勧め

    // 結果は、アルゴリズムで返される近似積分値を保持
    // abserr は |I - result| に対する上側境界の見積もり値
    // ここで I は積分値
    double result, abserr;

    // Nag_QuadProgress の構造、
    // max_num_subint 要素を持つ内部的に割り当てられたポインタを含む
    Nag_QuadProgress qp;
```

```
// NAG エラーパラメータ(構造)
NagError fail;

// パラメータは NAG の通信構造体で被積分関数に渡される
struct user param;
param.A  = 1.0;
param.Xc = 0.0;
param.W  = 1.0;

Nag_User comm;
comm.p = (Pointer)&param;

// 積分実行
// NAG の不定積分子で使用可能な 3 種類の不定境界タイプ
//  integrator Nag_LowerSemiInfinite, Nag_UpperSemiInfinite, Nag_Infinite
/*
d01smc(f_callback, Nag_LowerSemiInfinite, b, epsabs, epsrel, max_num_subint,
&result, &abserr, &qp, &comm, &fail);
*/
d01sjc(f_callback, a, b, epsabs, epsrel, max_num_subint,
&result, &abserr, &qp, &comm, &fail);

// エラーメッセージを出力してエラーをチェック
if (fail.code != NE_NOERROR)
    printf("%s¥n", fail.message);

// 次の 3 つのエラー以外のエラーは
// 入力が不正か、割り当ての失敗
```

```
    // 再び積分ルーチンを呼び出す前にメモリリークを避け
    // メモリ割り当てを解放する必要がある
    if (fail.code != NE_INT_ARG_LT && fail.code != NE_BAD_PARAM
        && fail.code != NE_ALLOC_FAIL)
    {
        NAG_FREE(qp.sub_int_beg_pts);
        NAG_FREE(qp.sub_int_end_pts);
        NAG_FREE(qp.sub_int_result);
        NAG_FREE(qp.sub_int_error);
    }

    printf("%g¥n", result);
}
```

高次積分関数

2 次以上の高次積分に対して、NAG 積分子関数 **d01wcc** を呼び出し、積分を実行します。

ユーザ定義のコールバック関数は、NAG の **d01wcc** 関数に渡されます。

```
double NAG_CALL f_callback(int n, double* z, Nag_User *comm)
{
        double tmp_pwr;
        tmp_pwr = z[1]+1.0+z[3];
        return z[0]*4.0*z[2]*z[2]*exp(z[0]*2.0*z[2])/(tmp_pwr*tmp_pwr);
}
```

メイン関数:

```
void nag_d01wcc_ex()
{
        // 入力変数
        int ndim = NDIM;  // 積分の次元
        double a[4], b[4];
```

```
        for(int ii=0; ii < 4; ++ii)  // 積分の間隔
        {
                a[ii] = 0.0;
                b[ii] = 1.0;
        }
        int minpts = 0;
        int maxpts = MAXPTS;  // 関数評価の最大数
        double eps = 0.0001; // 精度をセット

        // 出力変数
        double finval, acc;
        Nag_User comm;
        NagError fail;

        d01wcc(ndim, f_callback, a, b, &minpts, maxpts, eps, &finval, &acc,
        &comm, &fail);

        if (fail.code != NE_NOERROR)
                printf("%s¥n", fail.message);

        if (fail.code == NE_NOERROR || fail.code == NE_QUAD_MAX_INTEGRAND_EVAL)
        {
                printf("Requested accuracy =%12.2e¥n", eps);
                printf("Estimated value    =%12.4f¥n", finval);
                printf("Estimated accuracy =%12.2e¥n", acc);
        }
}
```

13.1.4 微分

ocmath_derivative 関数は、スムージング無しで単純な微分を計算するのに使用します。関数は、以下に示すように、ocmath.h で宣言されます。

```
int ocmath_derivative(

    const double* pXData, double* pYData, uint nSize, DWORD dwCntrl = 0);
```

関数は、すべての欠損値を無視し、データ点とその隣のデータ点間の 2 つの勾配の平均を取ることで微分を計算します。dwCntrl 引数が 0 というデフォルト値を使う場合、関数は、データの方向が変わるときに入力されます。

```
if( OE_NOERROR == ocmath_derivative(vx, vy, vx.GetSize()) )

    out_str("successfully");
```

dwCntrl が DERV_PEAK_AS_ZERO にセットされると、データの方向が変わった場合に関数に 0 が入力されます。

```
if( OE_NOERROR == ocmath_derivative(vx, vy, vx.GetSize(), DERV_PEAK_AS_ZERO) )

    out_str("successfully");
```

13.2 統計

ワークシート内で選択したデータ(列や行、ワークシート全体)の統計処理を行いたい場合があります。**データで操作する**: **数値データ**:**データ範囲**の章には、列/行のインデックスでデータ範囲を構築する方法があり、元のデータをベクターデータにします。

13.2.1 列と行の記述統計量

ocmath_basic_summary_stats 関数は、元データの合計数、平均値、標準偏差、歪度などの基本統計量を計算します。詳細は、Origin C ヘルプを参照してください。詳細は、Origin C ヘルプを参照してください。次の Origin C コードは、vData というベクターオブジェクトのデータのポイント数、平均、平均の標準誤差を計算し、出力します。

```
int N;

double Mean, SE;

ocmath_basic_summary_stats(vData.GetSize(), vData, &N, &Mean, NULL, &SE);

printf("N=%d¥nMean=%g¥nSE=%g¥n", N, Mean, SE);
```

13.2.2 度数カウント

ocmath_frequency_count 関数は、FreqCountOptions 構造体に従って度数カウントを計算するのに使用します。

```
// 度数カウントを行うソースデータ

vector vData = {0.11, 0.39, 0.43, 0.54, 0.68, 0.71, 0.86};



// ビンサイズ、設定の開始と終了などのオプションをセット

int nBinSize = 5;
```

```
FreqCountOptions fcoOptions;

fcoOptions.FromMin = 0;

fcoOptions.ToMax = 1;

fcoOptions.StepSize = nBinSize;

fcoOptions.IncludeLTMin = 0;

fcoOptions.IncludeGEMax = 0;



vector vBinCenters(nBinSize);

vector vAbsoluteCounts(nBinSize);

vector vCumulativeCounts(nBinSize);

int nOption = FC_NUMINTERVALS; // 最後のビンを拡張



int nRet = ocmath_frequency_count(

    vData, vData.GetSize(), &fcoOptions,

    vBinCenters, nBinSize, vAbsoluteCounts, nBinSize,

    vCumulativeCounts, nBinSize, nOption);



if( STATS_NO_ERROR == nRet )

    out_str("Done");
```

さらに、離散/カテゴリーデータに対して度数カウントを計算する 2 つの関数があります。1 つはテキストデータに対する ocu_discrete_frequencies で、もう 1 つは数値データに対する ocmath_discrete_frequencies です。また、2 次元データに対して度数カウントを計算する 2 つの関数があります。ocmath_2d_binning_stats と ocmath_2d_binning です。

13.2.3 相関係数

ocmath_corr_coeff 関数は、Pearson rank, Spearman rank, Kendall rank の相関係数を計算するのに使われます。

```
matrix mData = {{10,12,13,11}, {13,10,11,12}, {9,12,10,11}};

int nRows = mData.GetNumRows();

int nCols = mData.GetNumCols();



matrix mPeaCorr(nCols, nCols);
```

```
matrix mPeaSig(nCols, nCols);



matrix mSpeCorr(nCols, nCols);

matrix mSpeSig(nCols, nCols);



matrix mKenCorr(nCols, nCols);

matrix mKenSig(nCols, nCols);



if(STATS_NO_ERROR == ocmath_corr_coeff(nRows, nCols, mData, mPeaCorr, mPeaSig,

        mSpeCorr, mSpeSig, mKenCorr, mKenSig))

{

        out_str("Done");

}
```

13.2.4 正規性の検定

ocmath_shapiro_wilk_test** 関数を使って、Shapiro-Wilk の正規性の検定を実行します。ocmath_lilliefors_test** 関数を使って、Lilliefors の正規性の検定を実行します。***ocmath_kolmogorov_smirnov_test** 関数を使って、Kolmogorov-Smirnov の正規性の検定を実行します。

```
vector vTestData = {0.11, 0.39, 0.43, 0.54, 0.68, 0.71, 0.86};



NormTestResults SWRes;

if( STATS_NO_ERROR == ocmath_shapiro_wilk_test(vTestData.GetSize(), vTestData,

                &SWRes, 1) )

{

        printf("DOF=%d, TestStat=%g, Prob=%g¥n", SWRes.DOF, SWRes.TestStat, SWRes.Prob);

}
```

13.3 カーブフィッティング

13.3.1 線形フィット

Origin C で線形フィットのルーチンを実行するには、**ocmath_linear_fit** 関数を使用します。この関数では、重み付き線形フィットを実行し、パラメータ値や統計情報を含むフィット結果を取得できます。

以下の手順で、この関数を使用し、Origin C での線形フィットを実行して結果を指定したウィンドウとワークシートに出力します。

線形フィットを実行

線形フィットを実行する前に、データをインポートします。ここでは、独立、従属変数ともに 1 つずつ必要です。

Origin C のルーチンを始めます。3 つのステップが必要です。

1. c ファイルを作成し、以下ように空の関数を追加します。この関数に、次のステップからコードをコピーします。

```
#include <GetNBox.h> // used for GETN_ macros

void linearfit()

{

}
```

2. 線形フィットを実行するデータを取得します。独立/従属変数ともにベクトル変数を使用します。

```
// XY データをワークシートウィンドウから取得

Worksheet wks = Project.ActiveLayer();

if(!wks)

    return;  // データ付きワークシートをアクティブにする必要がある

WorksheetPage wp = wks.GetPage();



DataRange dr;

dr.Add("X", wks, 0, 0, -1, 0);  // x 列

dr.Add("Y", wks, 0, 1, -1, 1);  // y 列



vector vX;

dr.GetData(&vX, 0);  // x 列のデータを取得しベクトルへ



vector vY;

dr.GetData(&vY, 1);  // y 列のデータを取得しベクトルへ
```

3. GetN ダイアログを表示して、フィットオプションをコントロールし、ocmath_linear_fit 関数を呼び出して、このオプションで線形フィットします。

```
// GetN ダイアログでフィットオプションを表示するために GUI ツリーを用意
GETN_TREE(trGUI)
GETN_BEGIN_BRANCH(Fit, _L("Fit Options"))
GETN_ID_BRANCH(IDST_LR_OPTIONS) GETN_OPTION_BRANCH(GETNBRANCH_OPEN)
        GETN_CHECK(FixIntercept, _L("Fix Intercept"), 0)
                GETN_ID(IDE_LR_FIX_INTCPT)
        GETN_NUM(FixInterceptAt, _L("Fix Intercept at"), 0)
                GETN_ID(IDE_LR_FIX_INTCPT_AT)
        GETN_CHECK(FixSlope, _L("Fix Slope"), 0)
                GETN_ID(IDE_LR_FIX_SLOPE)
        GETN_NUM(FixSlopeAt, _L("Fix Slope at"), 1)
                GETN_ID(IDE_LR_FIX_SLOPE_AT)
        GETN_CHECK(UseReducedChiSq, STR_FITTING_CHECKBOX_USE_RED_CHI_SQR, 1)
                GETN_ID(IDE_FIT_REDUCED_CHISQR)
GETN_END_BRANCH(Fit)
if( !GetNBox(trGUI) )
{
        return; // Cancel ボタンが押される
}
LROptions stLROptions;
stLROptions = trGUI.Fit; // GUI ツリーから struct に値を割り当て


// 上の入力データとフィットオプションで線形フィットを実行
int nSize = vX.GetSize();  // data size
FitParameter psFitParameter[2];  // 2 つのパラメータ
```

```
RegStats stRegStats;  // フィットの統計

RegANOVA stRegANOVA;  // anova



int nRet = ocmath_linear_fit(vX, vY, nSize, psFitParameter, NULL,

                        0, &stLROptions, &stRegStats, &stRegANOVA);

if(nRet != STATS_NO_ERROR)

{

    out_str("Error");

    return;

}
```

結果をウィンドウに出力

計算が終了すると、フィット結果を特定のウィンドウに出力できます。ここで、パラメータの値は、ツリーとしてスクリプトウィンドウに出力し、統計情報は結果ログウィンドウに出力します。

```
void put_to_output_window(const FitParameter* psFitParameter,

                    const RegStats& stRegStats, const RegANOVA& stRegANOVA)

{

        // スクリプトウィンドウ、結果ログ、ワークシートに分析結果を出力

        // フィットパラメータをスクリプトウィンドウに印字

        vector<string> vsParams = {"Intercept", "Slope"};

        for(int iPara = 0; iPara < vsParams.GetSize(); iPara++)

        {

            printf("%s = %g¥n", vsParams[iPara], psFitParameter[iPara].Value);

        }



        // 統計結果を結果ログに出力

        Tree trResults;

        TreeNode trResult = trResults.AddNode("LinearFit");

        TreeNode trStats = trResult.AddNode("Stats");
```

```
        trStats += stRegStats;  // フィットの統計をツリーノードに追加



        TreeNode trANOVA = trResult.AddNode("ANOVA");

        trANOVA += stRegANOVA;  // anova 統計をツリーノードに追加



        string strResult;

        tree_to_str(trResult, strResult);  // ツリーを文字列に変換



        Project.OutStringToResultsLog(strResult);  // 結果ログに出力

}
```

結果をワークシートに出力

フィット結果を、指定したワークシートに出力することもできます。そして、ワークシートウィンドウ内の一般的な列フォーマットか、ツリー表示フォーマットで結果を出力できます。

以下の 2 つの方法は、**Datasheet::SetReportTree** メソッドを使用してツリー変数によるワークシートに結果を出力します。異なる点は、ワークシート作成時の WP_SHEET_HIERARCHY オプションで、2 番目の変数は **AddLayer** メソッドを使用しているのがわかります。

通常のワークシートに結果を出力

```
void output_to_wks(WorksheetPage wp, const FitParameter* psFitParameter)

{

        // レポートツリーを用意

        int nID = 100; // ノードは固有のノード ID が必要

        Tree tr;

        tr.Report.ID = nID++;

        TreeNode trReport = tr.Report;

        trReport.SetAttribute(TREE_Table, GETNBRANCH_TRANSPOSE);



        // 1 列

        trReport.P1.ID = nID++;

        trReport.P1.SetAttribute(STR_LABEL_ATTRIB, "Parameter"); // 列ラベル

        trReport.P1.SetAttribute(STR_COL_DESIGNATION_ATTRIB, OKDATAOBJ_DESIGNATION_X);
```

```
// 2 列
trReport.P2.ID = nID++;
trReport.P2.SetAttribute(STR_LABEL_ATTRIB, "Value"); // 列ラベル
trReport.P2.SetAttribute(STR_COL_DESIGNATION_ATTRIB, OKDATAOBJ_DESIGNATION_Y);

// 3 列
trReport.P3.ID = nID++;
trReport.P3.SetAttribute(STR_LABEL_ATTRIB, "Prob>|t|"); // 列ラベル
trReport.P3.SetAttribute(STR_COL_DESIGNATION_ATTRIB, OKDATAOBJ_DESIGNATION_Y);

// 表中に表示するためのベクトルを用意
vector<string> vsParamNames = {"Intercept", "Slope"};  // パラメータ名
vector vValues, vProbs;  // パラメータ値と prob
for(int nParam = 0; nParam < vsParamNames.GetSize(); nParam++)
{
    vValues.Add(psFitParameter[nParam].Value);
    vProbs.Add(psFitParameter[nParam].Prob);
}

// ツリーノードにベクトルを割り当て
trReport.P1.strVals = vsParamNames;
trReport.P2.dVals = vValues;
trReport.P3.dVals = vProbs;

// レポートツリーをワークシートに
int iLayer = wp.AddLayer("Linear Fit Params");
Worksheet wksResult = wp.Layers(iLayer);
```

```
        if(!wksResult.IsValid() || wksResult.SetReportTree(trReport) < 0)
        {
            printf("Fail to set report tree.¥n");
            return;
        }
        wksResult.AutoSize();
}
```

ツリー形式のワークシートに結果を出力

```
void output_to_tree_view_wks(WorksheetPage& wp, const RegStats& stRegStats)
{
        Tree tr;
        int nID = 100; // 各ノードは固有のノード ID が必要
        uint nTableFormat = GETNBRANCH_OPEN
                                        | GETNBRANCH_HIDE_COL_HEADINGS
                                        | GETNBRANCH_HIDE_ROW_HEADINGS
                                        | GETNBRANCH_FIT_COL_WIDTH
                                        | GETNBRANCH_FIT_ROW_HEIGHT;

        // ルートテーブルノードを用意
        tr.Report.ID = nID++; // Report ツリーノードを追加し、ノード ID を割り当て
        TreeNode trReport = tr.Report;
        // テーブルノードに表属性が必要
        trReport.SetAttribute(TREE_Table, nTableFormat);
        // ルートテーブルのタイトル
        trReport.SetAttribute(STR_LABEL_ATTRIB, "Linear Fit Stats");

        // 統計テーブルノードを用意
        trReport.Table.ID = nID++; // Table ツリーノードとノード ID を割り当て
```

```
TreeNode trTable = trReport.Table;

// テーブルノードに表属性が必要

trTable.SetAttribute(TREE_Table, nTableFormat|GETNBRANCH_TRANSPOSE);

// 統計表のタイトル

trTable.SetAttribute(STR_LABEL_ATTRIB, "Regression Statistics");


// 結果ノードを用意

trTable.Stats.ID = nID++; // Stats ツリーノードを追加してノード id を割り当て

TreeNode trStats = trTable.Stats;

trStats += stRegStats; // sturct から結果をツリーノードに追加をサポート


// ラベル設定。表の行ヘッダにこれらのテキストが表示

trStats.N.SetAttribute(STR_LABEL_ATTRIB, "Number of Points");

trStats.DOF.SetAttribute(STR_LABEL_ATTRIB, "Degrees of Freedom");

trStats.SSR.SetAttribute(STR_LABEL_ATTRIB, "Residual Sum of Squares");

trStats.AdjRSq.SetAttribute(STR_LABEL_ATTRIB, "Adj.R-Square");


// 他のノードを非表示にする

trStats.ReducedChiSq.Show = false;

trStats.Correlation.Show = false;

trStats.Rvalue.Show = false;

trStats.RSqCOD.Show = false;

trStats.RMSESD.Show = false;

trStats.NormResiduals.Show = false;


// 階層フォーマットとして新しいワークシートを作成するコントロール

DWORD   dwOptions = WP_SHEET_HIERARCHY | CREATE_NO_DEFAULT_TEMPLATE;

int iLayer = wp.AddLayer("Linear Fit Stats", dwOptions);
```

```
        Worksheet wksResult = wp.Layers(iLayer);
        if(!wksResult.IsValid() || wksResult.SetReportTree(trReport) < 0)
        {
            printf("Fail to set report tree.¥n");
            return;
        }
        wksResult.AutoSize();
}
```

13.3.2 多項式フィット

Origin C で多項式フィットのルーチンを実行するには、**ocmath_polynomial_fit** 関数を使用します。この関数では、重み付き多項式フィットを実行し、パラメータ値や統計情報を含むフィット結果を取得できます。

ここでは、この関数使用することによって Origin C で多項式フィットを実行する方法を紹介します。

多項式フィットを実行

多項式フィットを実行する前に、データをインポートします。ここでは、独立、従属変数ともに 1 つずつ必要です。

多項式フィットを実行する操作には、3 つのステップが必要です。

1.多項式フィットを実行するデータを取得します。独立/従属変数ともにベクトル変数を使用します。

```
Worksheet wks = Project.ActiveLayer();
if(!wks)
        return;  // 無効なワークシート

DataRange dr;
dr.Add("X", wks, 0, 0, -1, 0);  // x 列
dr.Add("Y", wks, 0, 1, -1, 1);  // y 列

vector vX, vY;
dr.GetData(&vX, 0);  // x 列データをベクトルに取得
dr.GetData(&vY, 1);  // y 列データをベクトルに取得
```

2.構造体変数を定義し、他のデータ型に関数のパラメータを渡します。これは、関数設定の初期化もできます。

```
// 結果出力のための構造体のみ定義

int nSize = vX.GetSize();

const int nOrder = 2;  // 順序



int nSizeFitParams = nOrder+1;

FitParameter psFitParameter[3];  // パラメータ数 = nOrder+1



RegStats psRegStats;  // フィット統計

RegANOVA psRegANOVA;  // anova 統計
```

3.引数を渡してデータに多項式フィットを実行します。

```
// デフォルトオプションである 2 次の多項式フィット

int nRet = ocmath_polynomial_fit(nSize, vX, vY, NULL, nOrder, NULL, psFitParameter,

                                                                    nSizeFitParams, &psRegStats,
&psRegANOVA);



// エラーのチェック

if(nRet!=STATS_NO_ERROR)

{

        out_str("Error");

        return;

}
```

結果を出力

計算が完了したら、スクリプトウィンドウ、結果ログ、ワークシートなどに結果を出力できます。

詳細は、**結果ウィンドウに出力ワークシートに出力** のセクションを確認してください（**分析とアプリケーション**:**カーブフィッティング**:**線形フィット**）

13.3.3 多重回帰

Origin は、**ocmath_multiple_linear_regression** 関数を使用して多重回帰を行います。この関数は、データの重みと線形回帰オプションを提供しています。この関数を実行した後、フィットパラメータや ANOVA 統計、推定の共分散と相関の行列といった詳細な出力が可能です。

次のセクションでは、この関数を使用して多重線形回帰を実行する方法を示します。

多重線形回帰を実行

多重線形回帰を実行する前に、データをインポートします。ここでは、3 つの独立変数と 1 つの従属変数が必要です。

1.多重線形回帰のためのデータをロードします。独立データyは、matrix に、従属データは vector に格納する必要があります。

```
// 1. 多重線形回帰のデータを取得

Worksheet wks = Project.ActiveLayer();

if( !wks )

    return; // データのワークシートがアクティブであることを確認



DataRange dr;

dr.Add("X", wks, 0, 0, -1, 2);  // 最初の 3 列

dr.Add("Y", wks, 0, 3, -1, 3);  // 4 列目



matrix mX;

dr.GetData(mX, 0, 0);  // 最初の 3 列のデータを matrix へ取得



vector vY;

dr.GetData(&vY, 1);  // 4 列目のデータを取得
```

2.パラメータ初期値を宣言して関数に渡します。

```
// 2. 入力と出力変数を用意

UINT nOSizeN = mX.GetNumRows();  // 観測数

UINT nVSizeM = mX.GetNumCols();  // 独立変数の数



LROptions stLROptions;  // 線形回帰オプションの設定のために使用

stLROptions.UseReducedChiSq = 1;



FitParameter stFitParameters[4]; // nVSizeM+1 にする

UINT nFitSize = nVSizeM+1;  // FitParameter のサイズ
```

```
RegStats stRegStats;  // フィットの統計取得のために使用

RegANOVA stRegANOV;  // anova 統計取得のために使用
```

3.用意したパラメータを関数に渡し、多重線形回帰を実行します。

```
// 3. 多重線形回帰を実行

// ここでは推定の共分散と相関の行列は取得せず、重みも使用しない

int nRet = ocmath_multiple_linear_regression(mX, nOSizeN, nVSizeM, vY, NULL,

                0, &stLROptions, stFitParameters, nFitSize, &stRegStats, &stRegANOV);

if( nRet != STATS_NO_ERROR )

{

    out_str("Error");

    return;

}
```

結果を出力

計算が完了したら、スクリプトウィンドウ、結果ログ、ワークシートなどに結果を出力できます。

詳細は、**結果ウィンドウに出力ワークシートに出力** のセクションを確認してください（ **分析とアプリケーション**:**カーブフィッティング**:**線形フィット** ）

13.3.4 非線形フィット

NLFit は Origin のバージョン 8 以降の機能です。この新しいフィット機構は、反復実行中にデータのコピーでフィットプロセスを扱います。新しいフィット機構はワークシートデータを直接繰り返しアクセスするので、古いフィット機構に比べ、早い操作を実行できます。

非線形曲線フィットでは、2 つのクラスを利用可能です。

NLFit

新しいフィットエンジンからの低いレベルの API を持つ Origin C クラスです。このクラスには、Origin のナレッジがなく、バッファ内のデータのコピーで動作します。このクラスを使用するためには、すべての必要なバッファ（ポインタ）を用意する必要があります。この区分けは、バックグラウンド処理としてフィットを実行する機能の開発のために用意されています。

NLFitSession

NLFit クラスを Origin オブジェクトに内包した、使いやすいインターフェースで、より高いレベルの Origin C クラスです。このクラスは、NLFit ダイアログ内のカーネルです。Origin とインターフェースで接続するための処理は難しく、NLFitSession はこの複雑性をうまく扱うので、Origin C コードではこのクラスを使用することをお勧めします。

非線形フィット

NLFitSession クラスを使用する前に、特定のヘッダファイルをインクルードする必要があります。

```
#include <..¥originlab¥NLFitSession.h>
```

また、*OriginC\Originlab\nlsf_utils.c* を現在のワークスペースに含め、コンパイルする必要があります。プログラムでファイルを追加するため、コマンドウィンドウまたはスクリプトファイルから下記の LabTalk コマンドを実行します。

```
Run.LoadOC(Originlab¥nlsf_utils.c, 16)
```

NLFitSession オブジェクトを定義し、フィット関数を Gauss にします。

```
// 関数を設定

NLFitSession nlfSession;

if ( !nlfSession.SetFunction("Gauss") )

{

        out_str("Fail to set function!");

        return;

}

// パラメータ名と数

vector<string>  vsParamNames;

int nNumParamsInFunction = nlfSession.GetParamNamesInFunction(vsParamNames);
```

DATA_MODE_GLOBAL モデル付きの 2 つの XY データセットをパラメータ共有のグローバルフィットを実行するためにセットします。

```
int nNumData = 2;

// 第 1 データセットをセット

if ( !nlfSession.SetData(vY1, vX1, NULL, 0, nNumData) )

{

        out_str("Fail to set data for the first dataset!");

        return;

}

// 第 2 データセットをセット

if ( !nlfSession.SetData(vY2, vX2, NULL, 1, nNumData, DATA_MODE_GLOBAL) )

{

        out_str("Fail to set data for the second dataset!");

        return;

}
```

パラメータ値を初期化するためのパラメータ初期化コードをセットします。

```
// パラメータ初期化

if ( !nlfSession.ParamsInitValues() )

{

        out_str("Fail to init parameters values!");

        return;

}
```

または、1 つずつパラメータ値を設定することもできます。

```
vector vParams(nNumParamsInFunction*nNumData);

// 第 1 データセットのパラメータ値をセット

vParams[0] = 5.5; // y0

vParams[1] = 26; // A

vParams[2] = 8; // xc

vParams[3] = 976; // w



// 第 2 データセットのパラメータ値をセット

vParams[4] = 2.3; // y0

vParams[5] = 26; // A

vParams[6] = 10.3; // xc

vParams[7] = 102; // w



int nRet = nlfSession.SetParamValues(vParams);

if(nRet != 0) // 0 はエラーなし

    return;
```

2 つのデータセットでパラメータ xc を共有します。

```
int nSharedParamIndex = 1; // 1, これは Gauss 関数での xc のインデックス

nlfSession.SetParamShare(nSharedParamIndex);
```

フィットを実行し、ステータスメッセージを出力します。

```
// フィット

int nFitOutcome;

nlfSession.Fit(&nFitOutcome);



string strOutcome = nlfSession.GetFitOutCome(nFitOutcome);

out_str("Outcome of the fitting session :" + strOutcome);
```

フィット統計結果を取得します。

```
int nDataIndex = 0;

RegStats        fitStats;

NLSFFitInfo     fitInfo;

nlfSession.GetFitResultsStats(&fitStats, &fitInfo, false, nDataIndex);

printf("# Iterations=%d, Reduced Chisqr=%g¥n", fitInfo.Iterations,

 fitStats.ReducedChiSq);
```

最終的なパラメータ値を取得します。

```
vector          vFittedParamValues, vErrors;

nlfSession.GetFitResultsParams(vFittedParamValues, vErrors);

// パラメータ xc は 2 つの入力データで共有

// そのため、xc の値は同じで、

// vParamValues には、1 回しか表示しない

// vsParamNames は Gauss 関数のパラメータ名が含まれる - y0, xc, w, A.

// 以下で xc 以外の 2 番目のデータセットのパラメータ名を追加

vsParamNames.Add("y0");

vsParamNames.Add("w");

vsParamNames.Add("A");

for( int nParam = 0; nParam < vFittedParamValues.GetSize(); nParam++)

{

    printf("%s = %f¥n", vsParamNames[nParam], vFittedParamValues[nParam]);

}
```

最終フィットパラメータを使用して、フィット曲線の Y 値を計算します。

```
vector vFitY1(vX1.GetSize()), vFitY2(vX2.GetSize());

// 第 1 データセットのためのフィット Y データセットを取得

nlfSession.GetYFromX(vX1, vFitY1, vX1.GetSize(), 0);

// 2 番目のデータセットのフィット Y を取得

nlfSession.GetYFromX(vX2, vFitY2, vX1.GetSize(), 1);
```

FDF ファイルにアクセスする

FDF ファイルに保存されているフィット関数設定はツリー変数にロードすることができます。*OriginC\system\FDFTree.h* ファイルをインクルードする必要があります。

```
#include <FDFTree.h>
```

そして、nlsf_FDF_to_tree 関数を使います。

```
string strFile = GetOpenBox("*.FDF");

Tree tr;

if(nlsf_FDF_to_tree(strFile, &tr))

{

        out_tree(tr);

}
```

13.3.5 XY 検索

独立変数から従属変数の値を取得するため、あるいは、従属変数から独立変数の値を取得するために、指定したパラメータ値を使用する方法を示しています。

線形

X から Y を取得する式は、

```
y = a + x * b;
```

Y から X を取得する式は、

```
x = (y - a) / b;
```

非線形

非線形関数の場合、NumericFunction クラスを使用して x から y を取得し、ocmath_find_xs 関数を使用して y から x を取得します。

X から Y を取得

```
#include <ONLSF.h>
#include <..¥Originlab¥nlsf_utils.h>
void _findy_from_x()
{
	// フィット関数オーガナイザダイアログのカテゴリに関連した
	// 適切な関数を使用する。F9 キーを押すとこのダイアログが開く。
	// Poly 関数は Polynomial カテゴリにある
	string  strFuncFileName = "Poly";

	Tree    trFF;
	if( !nlsf_load_fdf_tree(trFF, strFuncFileName) )
	{
		out_str("Fail to load function file to tree");
		return;
	}

	NumericFunction func;
	if (!func.SetTree(trFF))
	{
		out_str("NumericFunction object init failed");
		return;
	}

	int nNumParamsInFunc = trFF.GeneralInformation.NumberOfParameters.nVal;
	vector vParams(nNumParamsInFunc);
	vParams = NANUM;
	vParams[0] = 1;
```

```
        vParams[1] = 2;

        vParams[2] = 3;



        vector vX = {1, 1.5, 2};

        vector vY;

        vY = func.Evaluate(vX, vParams);

}
```

Y から X を取得

次の関数は、指定した Y 値から 2 つの X 値を取得する方法を示しています。実行前に、Samples\Curve Fitting\Gaussian.dat をワークシートにインポートし、このワークシートをアクティブにします。

```
#include <...¥originlab¥nlsf_utils.h>

#include <FDFTree.h>

void _findx_from_y()

{

        double y = 20; // X 値取得する Y 値は 20



        Worksheet wks = Project.ActiveLayer();

        if (!wks)

        {

                return;

        }



        //フィットするデータを取得

        DataRange dr;

        dr.Add(wks, 0, "X");

        dr.Add(wks, 1, "Y");

        DWORD dwPlotID;

        vector vDataX, vDataY;
```

```
        if(dr.GetData(DRR_GET_DEPENDENT | DRR_NO_FACTORS, 0, &dwPlotID, NULL,
        &vDataY, &vDataX) < 0)
        {
                printf("failed to get data");
                return;
        }

        uint nFindXNum = 2; //X の個数をセット
        vector vFindX;
        vFindX.SetSize(nFindXNum);

        string strFile = GetOriginPath() + "OriginC¥¥OriginLab¥¥nlsf_utils.c";
        PFN_STR_INT_DOUBLE_DOUBLE_DOUBLEP pFunc =
                Project.FindFunction("compute_y_by_x", strFile, true);
        string strFuncFileName = "Gauss";
        vector vParams(4);
        vParams[0] = 5.58333; // y0
        vParams[1] = 26; // xc
        vParams[2] = 8.66585; // w
        vParams[3] = 976.41667; // A

        int nRet = ocmath_find_xs(y, (uint)(vDataY.GetSize()), vDataX,
                vDataY, nFindXNum, vFindX, strFuncFileName, vParams.GetSize(),
                vParams, pFunc);

        if( OE_NOERROR == nRet )
                printf("Y = %g¥tX1 = %g¥tX2 = %g¥n", y, vFindX[0], vFindX[1]);
}
```

13.4 信号処理

Origin C は、信号処理、ノイズデータのスムージングからフーリエ変換 (FFT), 短時間 FFT(STFT), コンボリューション 、相関, FFT フィルタリング, ウェーブレット 分析に対するグローバル関数と NAG 関数のコレクションを提供しています。

Origin C 関数は、Origin C ヘルプ -> Origin C リファレンス -> グローバル関数 -> 信号処理カテゴリーで確認できます。

13.4.1 スムージング

ocmath_smooth 関数は、メディアンフィルタ、Savitzky-Golay スムージング、隣接平均スムージング の 3 つの方法をサポートしています。OriginC:Ocmath smooth (global function)

```
vector vSmooth; // 出力

vSmooth.SetSize(vSource.GetSize());



//Savitzky-Golay スムージングを実行。 Left=Right=7, quadratic

int nLeftpts = nRightpts = 3;

int nPolydeg = 2;

int nRet = ocmath_smooth(vSource.GetSize(), vSource, vSmooth, nLeftpts, SMOOTH_SG,

        EDGEPAD_NONE, nRightpts, nPolydeg);
```

13.4.2 FFT

fft_* 関数を使う前に、fft_utils.h をインクルードする必要があります。

```
#include <fft_utils.h>
```

FFT

fft_real は、離散フーリエ変換(FFT_FORWARD)または逆フーリエ変換(FFT_BACKWARD)を実行します。

```
fft_real(vSig.GetSize(), vSig, FFT_FORWARD); // エラーなしで 0 を返す
```

周波数スペクトル

fft_one_side_spectrum は、FFT 結果の片側スペクトルを計算するのに使用します。

```
fft_one_side_spectrum(vSig.GetSize(), vSig); // エラーなしで 0 を返す
```

IFFT

```
fft_real(vSig.GetSize(), vSig, FFT_BACKWARD); // エラーなしで 0 を返す
```

STFT

stft_real 関数は、1D の実数信号データに短時間 FFT を実行するのに使用します。**stft_complex** 関数は、1D の複素数信号データに短時間 FFT を実行するのに使用します。 以下は、実数データに対するサンプルです。

```
int nWinSize = 4;

vector win(nWinSize);

get_window_data(RECTANGLE_WIN, nWinSize, win);



matrix stft;

double stime, sfreq;

vector sig = {0, 0, 0, 1, 1, 0, 0, 0};

stft_real(sig, win, 0.1, 1, 4, stft, stime, sfreq);



for (int ii = 0; ii < stft.GetNumRows(); ii++)

{

    for (int jj = 0; jj < stft.GetNumCols(); jj++)

        printf ("%f¥t", stft[ii][jj]);

    printf ("¥n");

}
```

13.4.3 FFT フィルタ

Origin C は、FFT フィルタを実行するのに、ローパス、ハイパス、バンドパス、バンドブロック、しきい値、ローパスパラボリックなど複数のフィルタタイプをサポートしています。例えば、

```
double dFc = 6.5;

int iRet = fft_lowpass(vecSignal, dFc, &vecTime);
```

13.4.4 ウェーブレット分析

Origin C で、NAG 関数を呼び、ウェーブレット分析を実行できます。すべてのウェーブレット関数を見るには、Origin C ヘルプ -> Origin C リファレンス -> グローバル関数 -> NAG 関数 -> NAG 関数へのアクセスとヘルプ -> ウェーブレットのカテゴリ。関連のヘッダファイルをインクルードする必要があります。

```
#include <..¥OriginLab¥wavelet_utils.h>
```

次は、実数型の 1D 連続ウェーブレットのサンプルです。

```
int n = vX.GetSize();

int ns = vScales.GetSize();

matrix mCoefs(ns, n);
```

```
NagError fail;

nag_cwt_real(Nag_Morlet, 5, n, vX, ns, vScales, mCoefs, &fail);
```

13.5 ピークと基線

13.5.1 グラフまたはワークシートから入力 XY データを取得

以下のセクションでは、入力 XY データとしてワークシートまたはグラフから取得可能であることを示しています。ここをクリックしてウィンドウからデータを取得する方法のヘルプを確認できます。

13.5.2 基線を作成

ocmath_create_baseline_by_masking_peaks 関数は、正のピークのみ、負のピークのみ、正負両方のピークに対して基線を作成します。

次のサンプルは、入力 XY データ(vx, vy)で、正のピークと負のピークに対して基線を作成する方法を示します。

```
// 基線の XY ベクトルにメモリを確保

vector vxBaseline(vx.GetSize()), vyBaseline(vx.GetSize());



// 基線の XY データを見つける

int nRet = ocmath_create_baseline_by_masking_peaks(vx.GetSize(), vx, vy,

        vxBaseline.GetSize(), vxBaseline, vyBaseline, BOTH_DIRECTION);



// X データで XY データを昇順にソート

if( OE_NOERROR == nRet )

{

        vector<uint> vn;

        vxBaseline.Sort(SORT_ASCENDING, true, vn);

        vyBaseline.Reorder(vn);

}
```

13.5.3 基線を削除

基線の X 座標がピークの曲線の X 座標と同じであれば直接減算でき、それ以外の場合、基線を除去する前に補間を行う必要があります。次のコードは、補間を行う方法を示し、基線を除去します。現在のワークシートは、ピーク XY データと基線 XY データの 4 列を持っています。

```
Worksheet wks = Project.ActiveLayer();


Column colPeakX(wks, 0), colPeakY(wks, 1);
Column colBaseLineX(wks, 2), colBaseLineY(wks, 3);


// ピーク XY データを取得
// 基線を減算したいので参照で Y データを取得
vector vPeakX = colPeakX.GetDataObject();
vector& vPeakY = colPeakY.GetDataObject();


// 基線データの取得
vector vBaselineX = colBaseLineX.GetDataObject();
vector vBaselineY = colBaseLineY.GetDataObject();


if( vPeakX.GetSize() != vPeakY.GetSize()
        || vPeakX.GetSize() == 0
        || vBaselineX.GetSize() == 0
)
        return;


// 基線データの補間を行い、ピークデータと同じ X 座標を保持
vector vyBaseTemp(vPeakX.GetSize());
if(OE_NOERROR != ocmath_interpolate(vPeakX, vyBaseTemp, vPeakX.GetSize(),
        vBaselineX, vBaselineY, vBaselineX.GetSize(), INTERP_TYPE_LINEAR))
{
        return;
}
```

```
// 基線の減算

vPeakY -= vyBaseTemp;
```

13.5.4 ピークを検索

ocmath_find_peaks_* 関数は、複数の方法でピークを見つけるのに使われます。

次のサンプルは、nLocalPts で選択したローカルスコープの局所最大点を見つける方法を示します。nIndex で印を付けた現在のポイントに対して、スコープは [nIndex-nLocalPts, nIndex+nLocalPts]です。

```
// 出力ベクターとしてメモリを確保

UINT nDataSize = vxData.GetSize();

vector vxPeaks(nDataSize), vyPeaks(nDataSize);

vector<int> vnIndices(nDataSize);



// nDataSize は入力データ, vxData, vyData のサイズは

// 出力データ, ピークの数を返す

int nLocalPts = 10;

int nRet = ocmath_find_peaks_by_local_maximum( &nDataSize, vxData, vyData,

        vxPeaks, vyPeaks, vnIndices,

        POSITIVE_DIRECTION | NEGATIVE_DIRECTION, nLocalPts);



if(OE_NOERROR == nRet)

{

        printf("Peak Num=%d¥n", nDataSize);

        vxPeaks.SetSize(nDataSize);

        vyPeaks.SetSize(nDataSize);

}
```

Origin C は、次の 2 つの関数をサポートしています。**ocmath_test_peaks_by_height** と **ocmath_test_peaks_by_number** は、それぞれ指定した高さとピークの数によりピークを確認します。

次は、最小ピーク高さでピークを確認する方法を示すサンプルです。

```
// 元の Y データから最小値と最大値を取得

double dMin, dMax;

vyData.GetMinMax(dMin, dMax);
```

```
// 最高点、最低点からより大きな値を取得
// そして、20%を乗算し、ピークの最小高さを取得
double dTotalHeight = max(abs(dMax), abs(dMin));
double dPeakMinHeight = dTotalHeight * 20 / 100;

// 指定した最小高さでピークを確認
nRet = ocmath_test_peaks_by_height(&nDataSize, vxPeaks, vyPeaks, vnIndices,
        dPeakMinHeight);

printf("Peak Num = %d¥n", nDataSize);
for(int ii=0; ii<nDataSize; ii++)
{
        printf("Peak %d:(%f,%f)¥n", ii+1, vxPeaks[ii], vyPeaks[ii]);
}
```

13.5.5 ピークの積分をフィット

ピークの積分

ocmath_integrate 関数は、曲線以下の面積を積分するのに使われます。

次のサンプルは、1 つのピークの部分曲線の積分を実行します。

```
int i1 = 51, i2 = 134; // 1 つのピークの部分範囲を設定する開始と終了のインデックス
IntegrationResult IntResult; // 出力, 積分結果
vector vIntegral(i2+1); // 出力, 積分データ

// 積分し結果を出力
if( OE_NOERROR == ocmath_integrate(vx, vy, i1, i2, &IntResult, vIntegral,
MATHEMATICAL_AREA, NULL, false, SEARCH_FROM_PEAK) )
{
        printf("Peak 1:Peak Index = %d, Area = %g, FWHM = %g, Center = %g,
```

```
                Height = %g¥n", IntResult.iPeak, IntResult.Area, IntResult.dxPeak,

                IntResult.xPeak, IntResult.yPeak);

}
```

ピークをフィットする

Origin C の **NLFitSession** クラスは、異なるフィット関数でピークフィットを行うメソッドをサポートしています。

13.6 NAG 関数を使用する

13.6.1 ヘッダファイル

NAG 関数を呼ぶには、ヘッダーファイルまたは NAG 関数が宣言されているファイルをインクルードする必要があります。

すべての NAG ヘッダファイルを共通で使用する 1 つのヘッダファイルを下記に示します。通常、このヘッダファイルだけをコードにインクルードします。

```
#include <OC_nag.h> // 全てに共通の NAG ヘッダファイル
```

1 つまたは数個のみの NAG 関数が使われている場合、それぞれ個々の NAG ヘッダファイルをインクルードすることもできます。例えば、NAG 関数 f02abc がコード内で使われている場合、2 つの関連ヘッダファイルをインクルードする必要があります。

```
#include <NAG¥nag.h>    // NAG 構造体と型の定義

#include <NAG¥nagf02.h> // f02 関数の宣言も含む
```

13.6.2 エラー構造体

すべての NAG 関数は、NagError 構造体へのポインタの 1 つの引数を取ります。この構造体は、NAG 関数の実行が成功したかどうかをテストするために使われます。

下記のサンプルは、NAG 関数 *f02abc* がうまく動作するかどうかを示します。

```
NagError err;                    // エラー構造体を宣言

f02abc(n, mx, n, r, v, n, &err); // NAG f02abc 関数の呼び出し

if( err.code != NE_NOERROR )     // エラーが発生したら

    printf(err.message);         // エラーメッセージを出力
```

呼び出しが成功したかどうかを知る必要が無ければ、エラー構造体の宣言は必要ありません。そして **NAGERR_DEFAULT** マクロが代わりに渡されます。このマクロは NULL ポインタです。NAG 関数の将来のバージョンとの互換性を確実にするため、エラー構造体無しで操作できるなら、このマクロを使用した方がよいでしょう。

```
f02abc(n, mx, n, r, v, n, NAGERR_DEFAULT);
```

13.6.3 コールバック関数

NAG ライブラリで、ほとんどのルーチンはコールバック関数を含みます。コールバック関数を定義する前に、関数が呼ばれるときに NAG が期待している戻り型と引数の型を知る必要があります。

例えば、NAG 関数 *d01ajc* は次のようになります。ヘッダファイル *nagd01.h* で、最初の引数が **NAG_D01AJC_FUN f** であることが分かります。この引数はコールバック関数です。そして、*nag_types.h* で、**NAG_D01AJC_FUN** が **NAG_D01_FUN** の型であることがわかり、以下のように定義されます。

```
typedef double (NAG_CALL * NAG_D01_FUN)(double);
```

そして、次のようにコールバック関数を定義します。

```
double NAG_CALL myFunc(double x)

{

    double result;

    // 'x' に処理

    return result;

}
```

NAG 関数 *d01ajc* を呼び出すと、上述で定義した *myFunc* は、最初の引く数として渡されます。

c05adc を呼び出すサンプル

このサンプルは、NAG 関数 *c05adc* を呼び出す方法を示し、この 4 番目の引数がコールバック関数の引数です。**NAG_C05ADC_FUN** 型のこのコールバック関数は、*nag_types.h* で定義されます。

```
typedef double (NAG_CALL * NAG_C05ADC_FUN)(double);
```

定義から、戻り型と引数の型が double であることが分かります。そして、次のようにコールバック関数を定義できます。

```
double NAG_CALL myC05ADCfunc(double x)

{

    return exp(-x)-x;

}
```

以下のコードは、コールバック関数 *myC05ADCfunc* を渡して、関数 *c05adc* を呼び出す方法を示しています。

```
double a = 0.0, b = 1.0, x, ftol = 0.0, xtol = 1e-05;

NagError err;



c05adc(a, b, &x, myC05ADCfunc, xtol, ftol, &err);
```

13.6.4 NAG 関数は Origin からデータを取得する

多くの NAG 関数は、数値データの配列へのポインタを取ります。Origin のワークシートおよび行列シートは、そのデータのポインタを取得できます。このポインタは NAG 関数に渡すことができます。Origin C で、データは Dataset または DataRange オブジェクトを使って渡されます。 以下のセクションでは、Dataset および DataRange を使ってワークシートからデータを渡す方法を示しています。DataRange を使うことをお勧めします。

Dataset

Dataset が NAG 関数で期待されているデータ型であれば、Dataset オブジェクトを NAG 関数に渡すことができます。Origin ワークシート列のデータ型は、デフォルトで**文字と数値**です。すべてではありませんが、ほとんどの NAG 関数に対しては、NAG 関数が浮動小数点または整数型のポインタを期待しているので、このデータ型を渡すことができません。

Dataset が NAG 関数で期待されているデータ型であることが確実な場合、次のコードは Dataset オブジェクトを NAG 関数に渡すのに使うことができます。

```
// アクティブワークシートへのアクセス

Worksheet wks = Project.ActiveLayer();



// データセットを構築し、wks データにアクセス

Dataset dsX, dsY;

dsX.Attach(wks, 0);

dsY.Attach(wks, 1);



// NAG の nag_1d_spline_interpolant(e01bac) 関数を呼び出し

NagError err;

Nag_Spline spline;

e01bac(m, dsX, dsY, &spline, &err);
```

DataRange

DataRange クラスは、仮にワークシート列が**文字と数値**データ型であったとしても、ワークシートからのデータを vector に取得する GetData メソッドを提供します。GetData メソッドは、欠損値を持つ行を簡単に無視でき、これは NAG 関数にデータを渡すときに大変重要です。

DataRange を使って、Origin から NAG 関数にデータを渡すことが安全で、お勧めです。以下のサンプルは、それを行う方法を示しています。

```
void call_NAG_example()

{

        int i, numPoints = 5;



        // 新しいワークシートページを作成

        WorksheetPage pg;

        pg.Create("origin");
```

```
// アクティブワークシートにアクセスし 2 列追加
Worksheet wks = Project.ActiveLayer();
// X2 列を追加
i = wks.AddCol();
Column col(wks, i);
col.SetType(OKDATAOBJ_DESIGNATION_X);
// Y2 列を追加
wks.AddCol();

// 最初の 2 列でいくつかの開始 XY 値を作成
Dataset dsX, dsY;
dsX.Attach(wks, 0);
dsY.Attach(wks, 1);
for (i = 0; i < numPoints; i++)
{
        int r = rnd(0) * 10;
        if (r < 1)
                r = 1;
        if (i > 0)
                r += dsX[i - 1];
        dsX.Add(r);
        dsY.Add(rnd(0));
}

// データ範囲オブジェクトを作成
DataRange dr;
dr.Add(wks, 0, "X");
dr.Add(wks, 1, "Y");
```

```
// データ範囲を使用して wks からデータを vedtor にコピー
// このコピーは、欠損値を持つ行を無視
vector vX1, vY1;
dr.GetData(DRR_GET_DEPENDENT, 0, NULL, NULL, &vY1, &vX1);

// NAG を呼び出して係数を計算
NagError err;
Nag_Spline spline;
e01bac(vX1.GetSize(), vX1, vY1, &spline, &err);

// スプラインの XY 値を取得
vector vX2, vY2;
double fit, xarg;
for (i = 0; i < vX1.GetSize(); i++)
{
        vX2.Add(vX1[i]);
        vY2.Add(vY1[i]);
        if (i < vX1.GetSize() - 1)
        {
                xarg = (vX1[i] + vX1[i + 1]) * 0.5;
                e02bbc(xarg, &fit, &spline, &err);
                vX2.Add(xarg);
                vY2.Add(fit);
        }
}

// NAG で割り当てられたメモリを解放
```

```
        NAG_FREE(spline.lamda);

        NAG_FREE(spline.c);



        // ワークシートにスプライン値をコピー

        dsX.Attach(wks, 2);

        dsX = vX2;



        dsY.Attach(wks, 3);

        dsY = vY2;

}
```

13.6.5 NAG e04 関数の呼び出し方

NAG e04 関数は、関数の最小化や最大化のために主に使用されます。全ての e04 関数は、**Nag_E04_Opt** 構造体へのポインタであるパラメータが必要です。このような関数を実行した後、結果はデフォルトで windows のコンソールに出力されます。しかし、これは Origin で実行する場合安全ではありません。そのため、ここではデフォルトポインタである **E04_DEFAULT** を使用せず、**Nag_E04_Opt** 構造体変数の初期化を先に行い、NAG 関数に渡す前に出力ターゲットをファイルに変更します。

次のサンプルは、NAG 関数 **nag_opt_simplex** を安全に呼び出す方法を示します。そして、結果はファイルに出力します。

```
#include <OC_nag.h>

void text_e04ccc()

{

        double objf;

        double x[2];

        Integer n;



        printf("¥ne04ccc example:¥n");



        NagError fail;  // エラー

        Nag_E04_Opt opt;  // e04 のオプションパラメータ



        nag_opt_init(&opt);  // オプションパラメータ初期化, e04xxc = nag_opt_init
```

```
        // 出力ターゲットをファイルに変更
        strcpy(opt.outfile, "C:¥¥result.txt");

        n = 2;
        x[0] = 0.4;
        x[1] = -0.8;
        try
        {
                // NAG 関数を呼び出す, e04cccc = nag_opt_simplex
                nag_opt_simplex(n, funct, x, &objf, &opt, NAGCOMM_NULL, &fail);
        }
        catch(int err)
        {
                printf("¥nerror = %d¥n", err);  // 例外がある場合
        }
        printf("fail->code = %d¥n", fail.code);  // エラーコード
        printf("fail->message = %s¥n", fail.message);  // エラーメッセージ
}

// nag_opt_simplex のコールバック関数
void NAG_CALL funct(Integer n, double* xc, double* objf, Nag_Comm* comm)
{
        *objf = exp(xc[0])*(xc[0]*4.0*(xc[0]+xc[1])+xc[1]*2.0*(xc[1]+1.0)+1.0);
}
```

14 出力オブジェクト

14.1 結果ログ

結果ログは、出力の各ブロックごとに日時スタンプや結果に関連するウィンドウ名を自動的に出力するウィンドウです。ユーザインターフェースは、表示する結果を設定することができ、ウィンドウをフローティングにしたり、Origin のメインウィンドウにドッキングすることができます。

次のサンプルは、Project クラスの *OutStringToResultsLog* メソッドを使って Origin C から結果ログに出力する最も簡単なサンプルです。これは結果ログに出力する最も簡単なサンプルですが、最も制限があると考えることもできます。*OutStringToResultsLog* メソッドへの各呼び出しは、個別のログを考えることができ、現在の日時と関連ウィンドウを出力します。

```
string str = "Column1¥tColumn2¥tColumn3¥n3.05¥t17.22¥t35.48";

Project.OutStringToResultsLog(str);
```

14.2 スクリプトウィンドウ

スクリプトウィンドウは Origin C のデフォルトの出力ウィンドウです。文字列や数値を出力するときは、常にスクリプトウィンドウに表示されます。LabTalk の Type.Redirection プロパティを使って、出力されるウィンドウを変更することができます。このプロパティにより、アプリケーションの出力をスクリプトウィンドウ、コマンドウィンドウ、結果ログ、ノートウィンドウのにリダイレクトすることができます。詳細は LabTalk の *Type.Redirection* プロパティをご覧ください。

以下のサンプルは、現在の Redirection の設定を保存し、スクリプトウィンドウにリダイレクトするように設定し、次にコマンドウィンドウに出力するように変更し、そして保存した設定に戻します。

```
string strTypeRedir = "type.redirection";

double dCurTypeRedir;

LT_get_var(strTypeRedir , &dCurTypeRedir); // 現設定を取得



LT_set_var(strTypeRedir , 5); // スクリプトウィンドウは 5

out_str("Hello Script Window");



LT_set_var(strTypeRedir , 128); // コマンドウィンドウは 128

out_str("Hello Command Window");



LT_set_var(strTypeRedir , dCurTypeRedir); // 保存設定に戻す
```

14.3 ノートウィンドウ

最初のサンプルは、Text プロパティを使って Note ウィンドウのテキストで操作する方法を示しています。Text プロパティは、文字列に関するすべての設定を使用する string クラスです。

```
Note note;

note.Create(); // 目的の Note ウィンドウを作成

if( note )

{

    note.Text = "Hello Note window.";

    note.Text += "¥nAnother line of text."

}
```

次のサンプルは、Type.Redirection および Type.Notes$を使って、Origin C の出力をノートウィンドウにリダイレクトします。

```
Note note;

note.Create(); // 目的の Note ウィンドウを作成



LT_set_str("type.notes$", note.GetName());

LT_set_var("type.redirection", 2); // ノートウィンドウは 2



out_str("Hello Notes Window");
```

14.4 レポートシート

Datasheet クラスには、**GetReportTree** と **SetReportTree** メソッドがあり、ワークシートまたは行列シートにレポートを設定したり、そこから取得できます。

```
if( wks.SetReportTree(tr.MyReport) < 0 )

    out_str("Failed to set report sheet into worksheet.");



if( wks.GetReportTree(tr.MyReport) )

    out_tree(tr.MyReport);

else

    out_str("Failed to get report tree from worksheet.");
```

15 データベースへのアクセス

15.1 データベースからインポート

Origin C は、データベースからワークシートにデータをインポートする機能があります。以下のサンプルは、Origin の Samples サブフォルダにある Access データベースファイルをインポートすることで、これを行う方法を示しています。*ADODB.Reocrdset* オブジェクトは MSDN を参照することができます。接続文字列を作成する方法については、DbEdit X ファンクションを参照してください。

```
Object ocora;



try

{

        ocora = CreateObject("ADODB.Recordset");

}

catch(int nError)

{

        out_str("Failed to create ADODB.Recordset");

        return FALSE;

}



// Origin のサンプルフォルダから stars.mdb をインポート

string  strDatabaseFile = GetAppPath(1) +

    "Samples¥¥Import and Export¥¥stars.mdb";



// データベース接続文字列の準備

string strConn;

strConn.Format("Provider=Microsoft.Jet.OLEDB.4.0; Data Source=%s;

    User ID=admin; Password=;", strDatabaseFile);
```

```
// SQL 文字列を準備

string strQuery = "Select Stars.Index, Stars.Name, Stars.LightYears,

    Stars.Magnitude From Stars";




ocora.CursorLocation = adUseClient;

try

{

        ocora.open(strQuery, strConn, 1, 3);

}

catch(int nError)

{

        out_str("Failed to open Oracle database");

        return FALSE;

}



Worksheet wks;

wks.Create();



//ワークシートにデータを配置

BOOL                        bRet = wks.PutRecordset(ocora);

out_int("bRet = ", bRet);

return bRet;
```

15.2 データベースへのエクスポート

Origin C は、ワークシートのデータを指定したデータベースのテーブルにエクスポートする機能があります。次のステップは、フィットサマリーデータをデータベースにエクスポートする方法を示しています。

1. MySQL の"Analysis"というデータベースをセットアップし、それは"Lintilla"というコンピュータ上で動作しているものとします。

2. 9 つのフィールドを持つ"FittingSummary"というテーブルを作成し、最初の 2 つのフィールドのデータタイプを varchar(40)としてセットし、残りは double 型としてセットします。
3. OriginExe\Samples\Curve Fitting\autofit.ogw を開き、"Data"レイヤ上の列にデータを入力します。
4. 再計算の後、"Summary"レイヤをアクティブにし、次のコードを実行して、結果をデータベースにエクスポートします。

```
//データベースの設定に従ってユーザは接続とクエリ文字列を修正
//Server, Database, UID, PWD などの値がデータベースの設定
#define    STR_DB_CONN                              "Driver={MySQL ODBC 3.51 Driver};    ¥
               Server=Lintilla;Port=3306;Option=4;Database=Analysis;UID=test;PWD=test;"
#define    STR_QUERY                                "Select * from FittingSummary"

bool       write_wks_to_db()
{
        Worksheet wks = Project.ActiveLayer();
        if ( wks )
                return false;

        // "Lintilla"上の"Analysis"に接続
        string  strConn = STR_DB_CONN;
        string  strQuery = STR_QUERY;

        Object  oConn;
        oConn = CreateObject("ADODB.Connection");
        if ( !oConn )
                return error_report("Fail to create ADODB.Connection object!");
        oConn.Open(strConn);

        Object  oRecordset;
        oRecordset = CreateObject("ADODB.Recordset");
```

```
        if ( !oRecordset )
                return error_report("Fail to create ADODB.Recordset object!");

        // recordset を開く
        oRecordset.CursorLocation = 3; //adUseClient, 詳細は MSDN を参照
        oRecordset.Open(strQuery, oConn, 1, 3); //adOpenKeyset, adLockOptimistic

        int iRowBegin = 0, nRows = 8; //8 行
        int iColBegin = 0, nCols = 9; //9 列

        //LAYWKSETRECORDSET_APPEND は新しい recordset を追加;
        //LAYWKSETRECORDSET_REPLACE は既存の recordsets を置き換え
        int nOption = LAYWKSETRECORDSET_APPEND; //追加

        int nRet = wks.WriteRecordset(oRecordset, nOption,
                        iRowBegin, nRows, iColBegin, nCols);
        return (0 == nRet);
}
```

15.3 SQLite データベースへのアクセス SQLite

SQLite は、内蔵型で、サーバおよび設定不要な SQL データベースエンジンを組み込んだソフトウェアライブラリで、世界で最も幅広く使われている SQL データベースとされています。高度な機能があるので、SQLite はさまざまな業務分野やシステムで広く使われています。

SQLite3 は、最新のバージョンです。Origin は、Origin C から 32bit、64bit の SQLite データベースにアクセスする DLL を提供しています。SQLite3 API のプロトタイプを含むヘッダファイルを含める必要があります。

```
#include <oc_Sqlite.h>
```

これらの関数の使い方の簡単なサンプルがヘッダファイルの最後にあります。

Origin C は、SQLite へのアクセスをより簡単にするラッパークラス **OSQLite** も提供しています。この Origin C クラスを使うには、次のように、このクラスを含むヘッダファイルが含まれている必要があります。

```
//DataSet1.1.db はデータベースファイルで、これは Originlab という名前のテーブルを含む

//テーブルは次のステートメントで作成

//CREATE TABLE OriginLab(ID INTEGER NOT NULL, NUMBER INTEGER NOT NULL, SALARY INTE

//GER NOT NULL, Data BLOB NOT NULL);

#include <..¥Originlab¥oSQLite.h> //必要なヘッダファイル

#define          STR_DATABASE_FILE    "E:¥¥DataSet1.1.db"

#define          STR_QUERY_STRING     "select * from Originlab limit 80"

void test_OSQLite()

{

       OSQLite sqlObj(STR_DATABASE_FILE);

       LPCSTR lpSQL = STR_QUERY_STRING;

       sqlObj.Select(lpSQL);

       Worksheet wks;

       wks.Create("Origin");

       sqlObj.Import(wks);

       //データを修正した後、次のコードでデータをエクスポート

       //sqlObj.Export("OriginLab", wks);

}
```

16 LabTalk へのアクセス

16.1 LabTalk 変数の値を取得およびセットする

Origin C には、LabTalk の数値および文字列値を取得またはセットしたり、LabTalk スクリプトを実行する機能があります。

16.1.1 LabTalk の数値を取得およびセットする

Origin C のグローバル関数 **LT_get_var** および **LT_set_var** は、LabTalk の数値を取得およびセットするのに使うことができます。数値には、変数、システム変数、オブジェクトプロパティが含まれます。

```
double dOriginVer;

LT_get_var("@V", &dOriginVer);

printf("Running Origin version %f¥n", dOriginVer);
```

これは、データ表示ウィンドウで使用する最小のフォントサイズをセットする方法です。

```
LT_set_var("System.DataDisplay.MinFontSize", 12);
```

LabTalk の変数を一時的にセットし、操作を行い、LabTalk 変数を元の値に戻したい場合があります。これには主に 2 つの方法があります。最初の方法は、*LT_get_var* および *LT_set_var* を使って行う長いコードです。

```
double dProgressBar;

LT_get_var("@NPO", &dProgressBar); // 開始値の設定

LT_set_var("@NPO", 0); // 新しい値をセット



// 操作

LT_set_var("@NPO", dProgressBar); // 開始値に戻す
```

次の方法は、*LTVarTempChange* クラスを使う単純な方法です。クラスを使うには、単に変数名と一時的な値を渡すだけです。コンストラクタは開始値をデータメンバーに保存し、変数を一時的な値にセットします。デストラクタは変数をその開始値に戻します。

```
{

    LTVarTempChange progressBar("@NPO", 0);

    //

    // 操作

    //

}
```

16.1.2 LabTalk の文字列値を取得およびセットする

Origin C のグローバル関数 **LT_get_str** および **LT_set_str** は、LabTalk の文字列値を取得およびセットするのに使うことができます。文字列には、変数、文字列置換変数、オブジェクトプロパティが含まれます。

```
char szCustomDateFmt[200];

LT_get_str("System.Date.CustomFormat1$", szCustomDateFmt, 200);

printf("Custom Date Format 1:%s¥n", szCustomDateFmt);
```

これは、データ表示ウィンドウで使用するフォントをセットする方法です。

```
LT_set_str("System.DataDisplay.Font$", "Courier");
```

これはアクティブブックのアクティブシートの名前を変更する方法です。

```
LT_set_str("wks.name$", "MySheet");
```

16.2 LabTalk スクリプトを実行する

Origin C のグローバル変数 **LT_execute** は、文字列に保存されている LabTalk スクリプトを実行することができます。Format 文字列メソッドは、Origin C 変数を LabTalk スクリプトに渡すのに役立ちます。

```
string strScript;

string strBook = "Book1";

int iColStart = 2, iColEnd = 5;

strScript.Format("win -a %s;plotxy %u:%u;", strBook, iColStart, iColEnd);

LT_execute(strScript);
```

次のサンプルは、グローバル関数 LT_execute ではなく、Layer クラスの LT_execute メソッドを呼びます。グローバル関数 *LT_execute* を呼ぶと、スクリプトが実行され、スクリプトコードでレイヤが指定されていなければ、アクティブレイヤに対して操作されます。Layer クラスの *LT_execute* メソッドを呼ぶと、スクリプトが実行され、アクティブレイヤではなく、レイヤインスタンスに対して実行されます。

```
WorksheetPage wksPg("Book1");

Worksheet wks = wksPg.Layers(0);



WorksheetPage wksPgActive;

wksPgActive.Create("Origin"); // このページがアクティブ



LT_execute("wks.colWidth=16");     // アクティブレイヤの列幅をセット

wks.LT_execute("wks.colWidth=8"); // Book1 の列幅をセット
```

16.3 Origin C コードに LabTalk スクリプトを埋め込む

LT_execute は、文字列に含まれる LabTalk スクリプトを実行しますが、文字列に配置したくないような大きなブロックのスクリプトを実行したい場合があります。このようなときには、*_LT_Obj* ブロックを使うことができます。*_LT_Obj* ブロックは、大きな LabTalk スクリプトコードブロックを Origin C コードのフローに埋め込みます。LabTalk オブジェクトについての詳細は、LabTalk ヘルプ:LabTalk プログラミング:言語リファレンス:オブジェクトリファレンス

```
out_str("Choose an image file...");



_LT_Obj // LabTalk の FDlog を使ってファイルダイアログを表示

{

        // Origin C コード

        string strDefaultPath = GetOriginPath(); // Origin EXE パスを取得



        // FDLog オブジェクトにアクセスするための LabTalk スクリプト

        FDLog.Path$ = strDefaultPath;

    FDlog.UseGroup("image");

    FDlog.Open();

}



char szFileName[MAX_PATH];

LT_get_str("%A", szFileName, MAX_PATH);

printf("File Name:%s¥n", szFileName);
```

17 X ファンクションへのアクセス

Origin には、様々なタスクを処理するために、多くの X ファンクションが組み込まれています。X ファンクションは、LabTalk と Origin C 双方から呼び出せます。このセクションでは、Origin C の XFBase クラスを使用することで、Origin C から X ファンクションを呼び出す方法を紹介します。このメカニズムは、他の X ファンクション内の 1 つの X ファンクションを呼び出すのにも使用できます。

XFBase クラスは、Origin C システムフォルダ内にある XFBase.h ヘッダファイルで実行されます。XFBase.h ヘッダファイルは、Origin.h ヘッダファイルに含まれていないので、XFBase クラスの使用のために個別に含める必要があります。

```
#include <XFBase.h>
```

17.1 X ファンクション impFile を Origin C から呼び出す

Origin C 内で、X ファンクションを使用するための手順を以下に示します。

1. 特定の X ファンクションから構成されたオブジェクトを宣言
2. X ファンクションオブジェクトから、SetArg オプションを使用して引数を割り当てる
3. X ファンクションオブジェクトの Evaluate メソッドを使用して X ファンクションを実行

次の Origin C コードは、Origin にファイルをインポートするための一般的な関数を定義します。この関数は、データファイル名とインポートフィルタ名の 2 つの引数をもちます。**impFile** X ファンクションに対する他の引数は、デフォルトの値を使用します。

```
bool call_impFile_XF(LPCSTR lpcszDataFile, LPCSTR lpcszFilterFile)
{
    string strDataFile = lpcszDataFile;
    string strFilterFile = lpcszFilterFile;

    // X ファンクション名を使用して XFBase のインスタンスを作成
    XFBase xf("impFile");
    if (!xf)
        return false;

    // 引数 'fname' をセット
    if (!xf.SetArg("fname", strDataFile))
        return false;
```

```
    // 引数 'filtername' をセット
    if (!xf.SetArg("filtername", strFilterFile))
        return false;

    // X ファンクション実行のために、XFBase の 'Evaluate' メソッドを呼び出す
    if (!xf.Evaluate())
        return false;

    return true;
}
```

次の Origin C コードは、上で定義した関数 call_impFile_XF を呼び出し、画像ファイルのインポートに使用する方法を示します。

```
// Origin のサンプルフォルダにあるビットマップファイル Car インポートします。
string strImageFile = GetAppPath(TRUE) +
    "Samples¥¥Image Processing and Analysis¥¥Car.bmp";

// 画像インポートフィルタを使用してビットマップ画像をインポート
string strFilterFile = GetAppPath(TRUE) + "Filters¥¥Image.oif";

// X ファンクション impFile で使用するために関数を呼び出す
call_impFile_XF(strImageFile, strFilterFile);
```

Note:

1. X ファンクションへのアクセスについての詳細情報は、次のヘルプファイルをご覧ください。**ヘルプ:プログラミング > Origin C > Examples > Accessing X-Functions > Accessing X-Function**
2. X ファンクションについての詳細情報は、次のヘルプファイルをご覧ください。**ヘルプ:X ファンクション**

18 ユーザインターフェース

ここでは、Origin C 関数がユーザからの入力を受け付ける方法を説明します。

18.1 ダイアログ

Origin C 関数でダイアログを作成する方法を紹介しています。

18.1.1 組み込みダイアログボックス Dialog Boxes

入力ボックス

入力ボックスはプログラムのユーザからテキスト形式の情報を求めます。入力ボックスを開くには、グローバル関数 *InputBox* が使われます。

```
// string を入力

string strName = InputBox("Please enter your name", "");

printf("Name is %s.¥n", strName);



// 数値を入力

double dVal = InputBox(0, "Please enter a value");

printf("Value is %g.¥n", dVal);
```

メッセージボックス

メッセージボックスは、情報を表示したり、ユーザに選択を促すために使われます。表示する情報は、操作を続ける前にユーザに注意を促すための重要なものにします。

最初のサンプルは、OK ボタンだけを持つ単純なメッセージボックスで、ファイルのダウンロードが成功したことを通知します。

```
string strTitle = "File Download";

string strMsg = "Your file downloaded successfully.";

MessageBox(GetWindow(), strMsg, strTitle, MB_OK);
```

次のサンプルは、OK とキャンセルボタンを持つ感嘆符のアイコン付きメッセージボックスで、ユーザに操作を元に戻すことができないという注意を促します。ユーザは操作を続行するか、キャンセルするかを選択できます。

```
string strTitle = "Delete Data";

string strMsg = "You will not be able to undo this change.";

int nMB = MB_OKCANCEL|MB_ICONEXCLAMATION;
```

```
if( IDOK == MessageBox(GetWindow(), strMsg, strTitle, nMB) )

    out_str("Data has been deleted");
```

次の例は、はいーいいえのボタンを持つ疑問符のアイコン付きのメッセージボックスです。これは、ユーザに操作を続行するかどうかを尋ねるのに使用しています。

```
string strTitle = "Close Windows";

string strMsg = "Are you sure you want to close all windows?";

int nMB = MB_YESNO|MB_ICONQUESTION;

if( IDYES == MessageBox(GetWindow(), strMsg, strTitle, nMB) )

    out_str("All windows have been closed.");
```

プログレスボックス

プログレスボックスは、ソフトウェアがデータを処理中であることを示す小さなダイアログボックスです。このダイアログボックスは、処理の進行状況の割合を表示するプログレスバーを含みます。プログレスダイアログは、通常、反復ループに使用します。

```
int iMax = 10, iMin = 0;

progressBox prgbBox("This is a ProgressBox example:");

prgbBox.SetRange(iMin, iMax);



for (int ii=iMin; ii<=iMax; ii++)

{

    if(prgbBox.Set(ii))

        printf("Hi, it is now at %d.¥n", ii);

    else

    {

        out_str("User abort!"); // Cancel ボタンをクリックして停止

        break;

    }

    LT_execute("sec -p 0.5");

}
```

ファイルダイアログ

Origin C は、すべてに共通のファイルダイアログの関数を提供しています。これには、1 つのファイルを開く、複数ファイルを開く、ファイルを保存する、フォルダを選択するダイアログが含まれます。次のセクションは、自分自身のアプリケーションでこれらのダイアログを使用する方法を示します。

ファイルを開くダイアログ

```
StringArray saFiletypes(3);

saFiletypes[0]="[Project (*.OPJ)] *.OPJ";

saFiletypes[1]="[Old version (*.ORG)] *.ORG";

saFiletypes[2]="[Worksheets (*.OGW)] *.OGW";



string strPath = GetOpenBox( saFiletypes, GetAppPath(false) );

out_str(strPath);
```

複数ファイルを開くダイアログ

```
StringArray saFilePaths;

StringArray saFileTypes(3);

saFileTypes[0]="[Project (*.OPJ)] *.OPJ";

saFileTypes[1]="[Old version (*.ORG)] *.ORG";

saFileTypes[2]="[Worksheets (*.OGW)] *.OGW";



// Ctrl または Shirt キーで複数ファイルを選択

int iNumSelFiles = GetMultiOpenBox(saFilePaths, saFileTypes, GetAppPath(false));
```

ファイルを保存ダイアログ

```
string strDefaultFilename = "Origin";

FDLogUseGroup nFDLogUseGroup = FDLOG_ASCII; // ASCII ファイルグループ



string strPath = GetSaveAsBox(nFDLogUseGroup,GetAppPath(false),strDefaultFilename);

out_str( strPath );
```

パスブラウザダイアログ

```
string strPath = BrowseGetPath(GetAppPath() + "OriginC¥¥", "This is an example");

out_str(strPath);
```

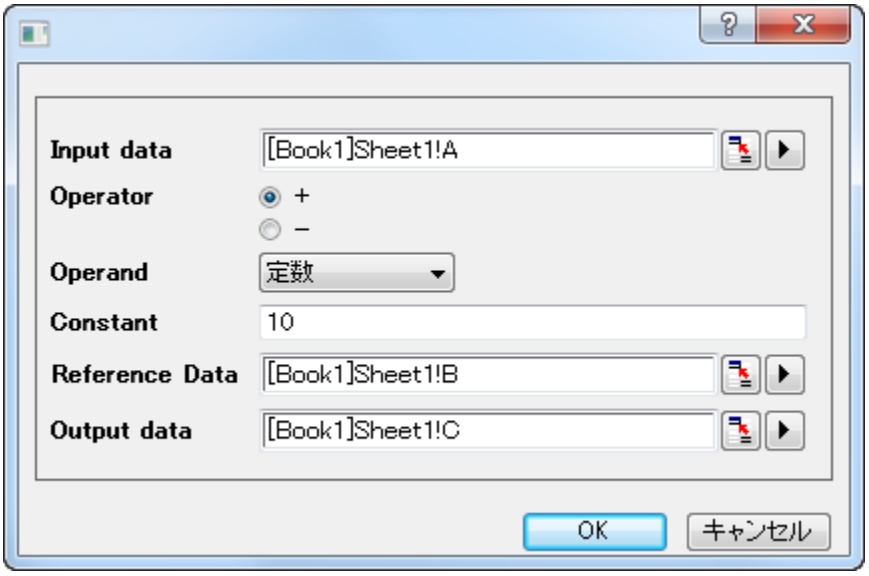
18.1.2 GetN ダイアログ

GetN Simple Dialog

シンプルなダイアログ

GetN マクロと GetNBox 関数を使用して、シンプルなダイアログを作成できます。

ダイアログは、下図のようなものです。



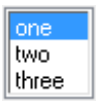
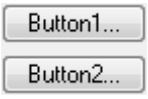
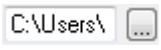
次の関数を実行して上のダイアログを開きます。

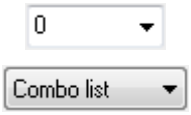
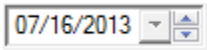
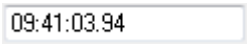
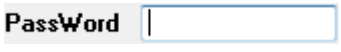
```
#include <GetNbox.h>

void simple_dialog()

{

    GETN_BOX(trRoot)    // "trRoot" というツリー変数を定義



    // データ範囲のコントロール

    GETN_INTERACTIVE(Input, "Input data", "[Book1]Sheet1!A")



    // ラジオボタン

    GETN_RADIO_INDEX_EX(Operator, "Operator", 0, "Add|Subtract")
```

```
    // リストボックス

    GETN_LIST(type, "Operand", 0, "Constant|Reference Data")



    // 文字列編集ボックス

    GETN_STR(Constant, "Constant", "10")



    // 参照データのデータ範囲を選択

    GETN_INTERACTIVE(Reference, "Reference Data", "[Book1]Sheet1!B")



    // 出力データのための列を選択

    GETN_INTERACTIVE(Output, "Output data", "[Book1]Sheet1!C")



    // ダイアログを開く

    GetNBox(trRoot );

}
```

コントロール

次表に、一般的なコントロールを示します。他のコントロールと、スタイルの設定については、Origin C Reference:Macros:GetN を確認してください。

画像	名前
	• GETN_RADIO_INDEX • GETN_RADIO_INDEX_EX
	• GETN_CHECK
	• GETN_LISTBOX • GETN_MULTISEL_LISTBOX
	• GETN_BUTTON_GROUP
	• GETN_BUTTON

	• GETN_COMBO(数値) • GETN_LIST(文字列。選択のインデックスを返す) • GETN_STRLIST(文字列。選択されたテキストを返す) • GETN_STR_GROUP(複数選択)
	• GETN_COMBO_BUTTON
	• GETN_RANGE
	• GETN_STR(文字列) • GETN_NUM(数値) • GETN_MULTILINE_TEXT(複数行テキスト)
	• GETN_SPINNOR_DOUBLE
	• GETN_SLIDER • GETN_SLIDEREDIT(編集可)
	• GETN_COLOR
	• GETN_SYMBOL
	• GETN_SEPARATOR_LINE
	• GETN_DATE
	• GETN_TIME
	• GETN_PASSWORD
	• GETN_XYRANGE • GETN_XYRANGE_COMPLEX • GETN_XYZRANGE • GETN_INTERACTIVE

イベントハンドラー

上のダイアログで、ダイナミックな表示/非表示の制御や、コンボリストの構築などのために、*node_event* 関数を追加できます。

以下のサンプル関数

```
GetNBox(trRoot);
```

を次のようにします。

```
GetNBox(trRoot, node_event);
```

以下のようにイベント関数を追加します。

```
int node_event(TreeNode& trRoot, int nRow, int nEvent, DWORD& dwEnables,
        LPCSTR lpcszNodeName, WndContainer& getNContainer, string& strAux,
        string& strErrMsg)
{
    if( 0 == lstrcmp(lpcszNodeName, "type") || GETNE_ON_VALUE_CHANGE == nEvent
    || GETNE_ON_INIT == nEvent )
    {
        trRoot.Constant.Show = (0 == trRoot.type.nVal); // Constant を表示
        trRoot.Reference.Show = (1 == trRoot.type.nVal); // 参照を表示
    }
    return 0;
}
```

適用ボタン

デフォルトの GetN ダイアログには、OK とキャンセルボタンがありますが、適用ボタンはオプションです。適用ボタンが表示されていて、ユーザがこのボタンをクリックすると、ある操作のためのイベント関数を呼び出すことができます。

次のサンプルは、GetN ダイアログに適用ボタンを追加する方法と、適用ボタンをクリックした際にイベント関数 *_apply_event* を呼び出す方法を示します。

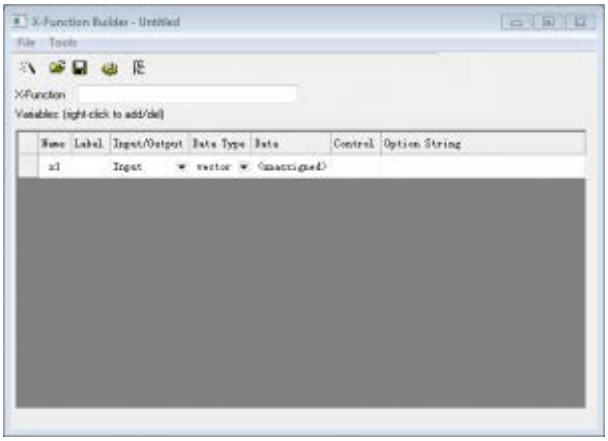
```
#include <GetNbox.h>
void GETN_Apply_ex1()
{
    GETN_TREE(tr)
    GETN_COLOR(LineColor, "Color", 3)
```

```
    // カスタムパネルを含めるためのカラーリストをセットするオプション
    GETN_COLOR_CHOICE_OPTIONS(COLORLIST_CUSTOM | COLORLIST_SINGLE)

    bool bShowApply = true;
    if(GetNBox(tr, NULL, "Example", NULL, GetWindow(), bShowApply,
                _apply_event))
    {
        out_str("Click OK");
    }
}

// 適用ボタンイベント関数のインターフェースは、
// PAPPLY_FUNC typedef を参照
bool _apply_event(TreeNode& tr)
{
    int nIndex = tr.LineColor.nVal;
    UINT cr = color_index_to_rgb(nIndex);
    printf("Red = %d, Green = %d, Blue = %d¥n", GetRValue(cr),
                GetGValue(cr), GetBValue(cr));
    return true;
}
```

18.1.3 X ファンクション

X ファンクションでダイアログを作成

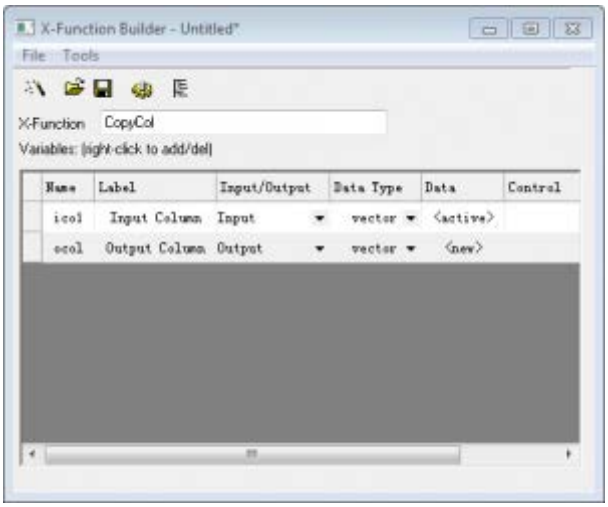
このサンプルでは、X ファンクションビルダを使って X ファンクションを作成することにより、ダイアログ生成の自動化について説明します。X ファンクションビルダはツールメニューで「X ファンクション・ビルダ」を選択するか、F10 を押すと開きます。



X ファンクションの作成

以下のステップは X ファンクションの作成手順を示します。今回の例では、1 つの列のデータを他の列にコピーするタスクを担う X ファンクションを作成します。

1. X ファンクションビルダを開き、X ファンクションの名前を「CopyCol」とし、変数リスト上で右クリックして「変数の追加」を選択して 2 つ目の変数を追加します。変数の名前、ラベル、他の値を次の画像と同じになるように入力します。



2.変数に必要な変更を加えたら、OXF ファイルを保存するボタン をクリックして X ファンクションを保存します。名前を付けて保存ダイアログが開かれたら、保存ボタンをクリックします。

3.では、目的の動作をするための Origin C コードを入力して X ファンクションを作成しましょう。コードビルダボタン をクリックします。これは X ファンクションをコードビルダで開き、Origin C コードを入力出来る状態にします。メインの関数に以下の Origin C コードを追加します。

```
ocol = icol;
```

4.コンパイルボタンをクリックしてからダイアログに戻るをクリックすると X ファンクションビルダに戻ります。OXF ファイルの保存ボタンをクリックしましょう。

X ファンクションダイアログを開きます。

X ファンクションをすぐにテストできます:

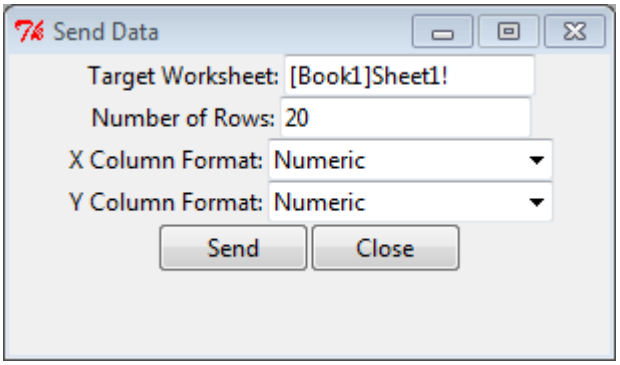
1. 2 つの列がある、新しいワークシートを作成します。列 A に行番号を入力し、ヘッダをクリックして列全体を選択します。
2. スクリプトまたはコマンドウィンドウに「CopyCol -d」と入力し(カギ括弧は入力しないでください)、Enter を押します。
3. ダイアログが開いたらデフォルトの値のまま OK ボタンをクリックします。

X ファンクションが実行されると、ワークシートは 3 番目の列を追加し、そこに列 A のデータをコピーします。

18.1.4 Python ダイアログ

Python で作成したダイアログ

このサンプル (\samples\Python\Run SendData toWks py.opj) では、Python の **tkinter** モジュールを使っインタラクティブなグラフィックインターフェースを作成し、データをインポートする前にデータタイプ、ターゲットとするワークシート、行番号を指定するを示します。Python コードを表示するには、**コードビルダ**を開く(**表示:コードビルダ**または **ALT+4**)か、または、**ワークスペース**パネルの**プロジェクト**フォルダをブラウズして、添付の Python ファイルにアクセスします。



18.1.5 ダイアログビルダ

ダイアログビルダは、OriginC で Microsoft Visual C++ のリソースを使い、Origin 内で使用するフローティングツール、ダイアログボックス、ウィザードを作成するものです。ウィンドウ共通のコントロール、Origin のワークシート、グラフコントロールなどの全てのリソース要素は Origin C から制御できます。これは、ダイアログビルダライセンスが必要でしたが、Origin 8.5 以降この制約はなくなり、全ての Origin で使用できるようになりました。

このガイドでは、Microsoft Visual C++ を使用して、ダイアログを含むリソースのみの DLL を作成し、ダイアログを表示するために Origin C を使用する方法を紹介したチュートリアルを含みます。リソースのみの DLL を作成して、Origin C からそのリソースにアクセスする方法の詳細は、詳しい説明が追加のセクションにあります。

ダイアログビルダサンプル

リソース DLL を含む、ダイアログビルダのサンプルファイルは Origin のインストールフォルダの*\Samples\Dialog Builder* サブフォルダにあります。

単純な Hello World ダイアログ

Hello World DialogDialog Builder

VC でリソース DLL を作成する

Origin Dialog AppWizard で作成する

1. Visual C++ 6.0 を開始し、ファイル->新規を選択し、新しいダイアログを開きます。プロジェクトタブで、Origin Dialog AppWizard を選択し、プロジェクト名を"ODialog"にセットし、場所を選択して、OK をクリックします。
2. シンプルダイアログを選び、次へをクリックします。

3. Origin C を選択したまま、完了をクリックし、OK をクリックします。1 つのシンプルダイアログを持つリソースファイルとそれに関連したソースファイルおよびヘッダファイルが生成されます。
4. ビルド->アクティブな構成の設定メニューを選択し、デバックまたはリリースを選びます。
5. ビルド->ODialog.dll のビルドを選択し、DLL を作成します。
6. 上記で指定したファイルの場所に移動します。DLL ファイルを Debug または Release フォルダの外側にコピーし、DLL ファイルのパスを ODialog.cpp ファイルのパスと同じにします。
7. Origin C のコードビルダで ODialog.cpp ファイルを開き、コンパイルし、DoMyDialog 関数を実行して、ダイアログを開きます。

Win32 ダイナミックリンクライブラリで作成する

このセクションは、Visual C++ 6.0 でリソースのみの DLL を作成する方法について説明しています。

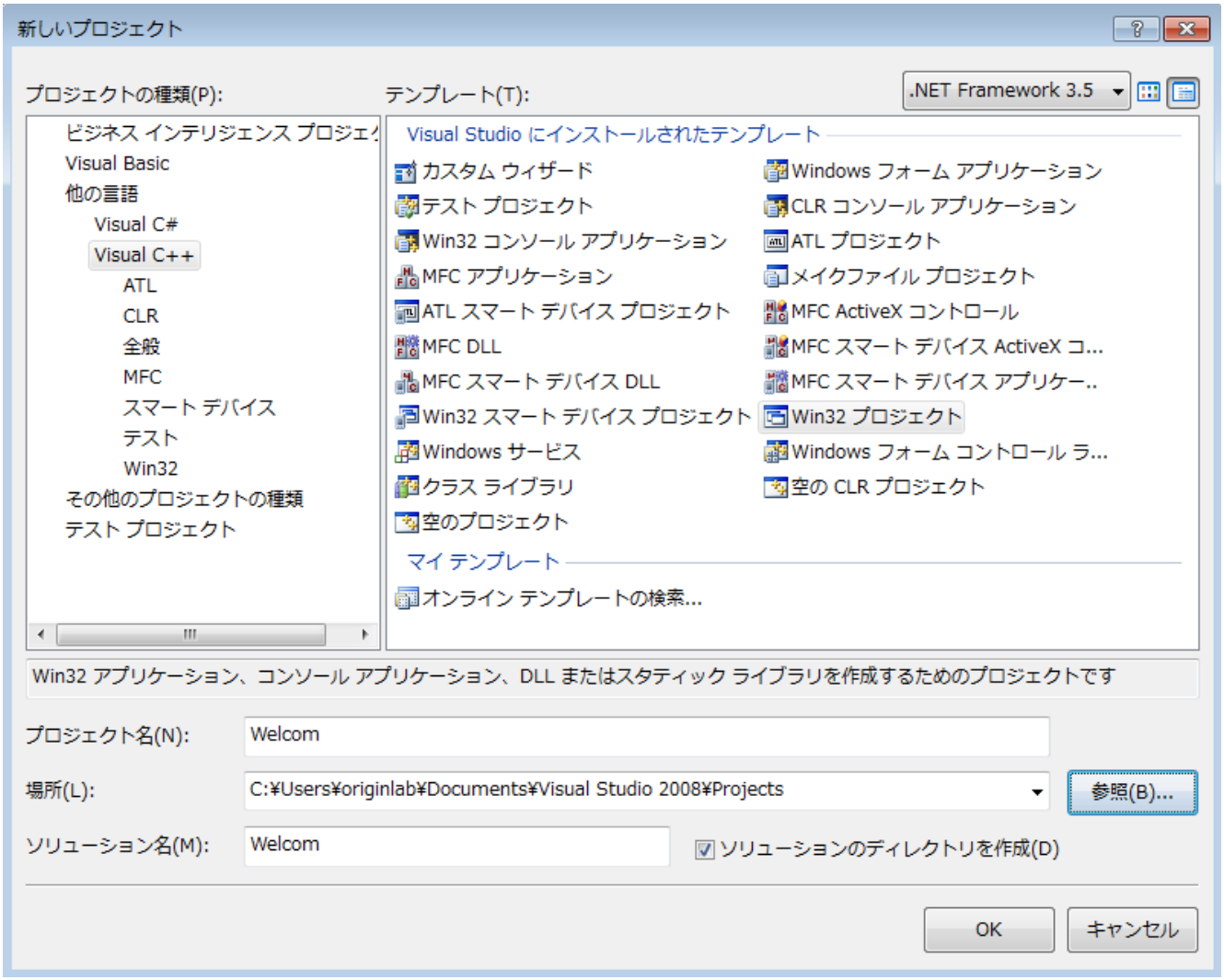
1. Visual C++ 6.0 を開始し、ファイル->新規を選択し、新しいダイアログを開きます。プロジェクトタブで、プロジェクトテンプレートとして Win32 ダイナミックリンクライブラリを選択し、プロジェクト名を ODialog にセットし、場所を選択して、OK をクリックします。現れたダイアログで、シンプル DLL プロジェクトを選び、完了をクリックします。
2. プロジェクト->設定を選び、プロジェクト設定ダイアログを開きます。リソースタブで、ODialog.res のようなリソースファイル名をセットし、ソフトウェア設定に従って言語を選択し、OK をクリックします。
3. 挿入->リソースを選び、プロジェクトにリソースを挿入します。ダイアログとそのコントロールに対して、ダイアログ ID を IDD_OC_DIALOG にセットします。
4. ファイル->名前を付けて保存を選び、リソーススクリプトを ODialog.rc として保存します。プロジェクト->プロジェクトに追加->ファイルを選び、ODialog.rc ファイルを選び、プロジェクトに追加します。
5. 言語が英語でなければ、このステップを行っいます。ワークスペースビューのリソースタブで、リストツリーを開き、IDD_OC_DIALOG を右クリックし、プロパティを選び、ダイアログで言語をニュートラルを選びます。
6. デバッグまたはリリースの構成で、プロジェクト全体をビルドします。結果の DLL ファイルが、Debug または Release サブフォルダに生成されます。

Visual Studio 2008 でリソースのみの DLL を作成する

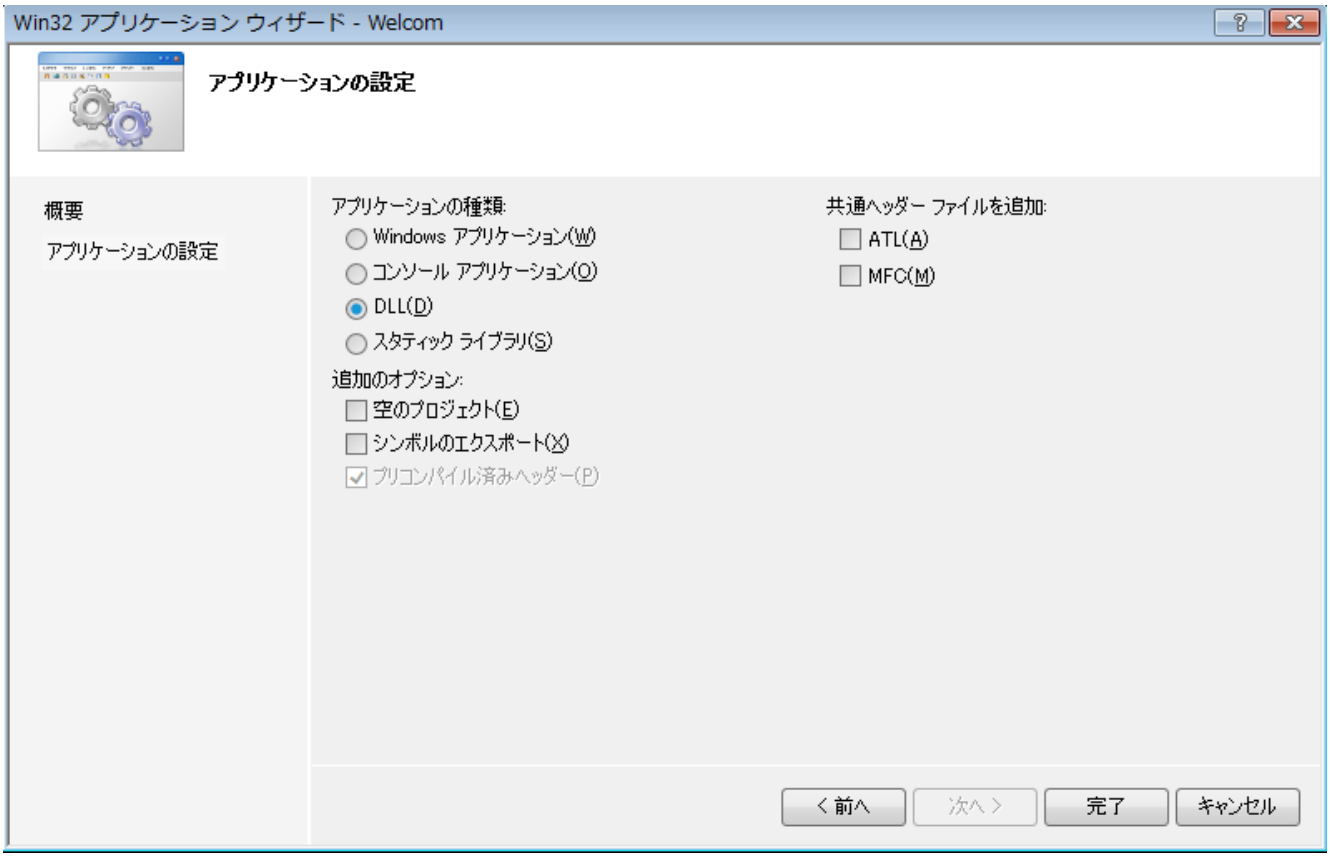
Visual Studio 2008 でリソースのみの DLL を作成する一般的な処理について説明しています。以下のステップは、Origin から Origin C を使ってアクセスするリソースのみの DLL を VS2008 で作成する方法を示しています。

1. Microsoft Visual Studio 2008 を起動します。
2. ファイル->新規->プロジェクトを選択し、新しいプロジェクトを作成します。

3. 新しいプロジェクトダイアログで、プログラム言語として Visual C++を選び、テンプレートとして Win32 プロジェクトを選び、プロジェクト名には"Welcome"とし、その場所を下記のように選択して、OK をクリックします。



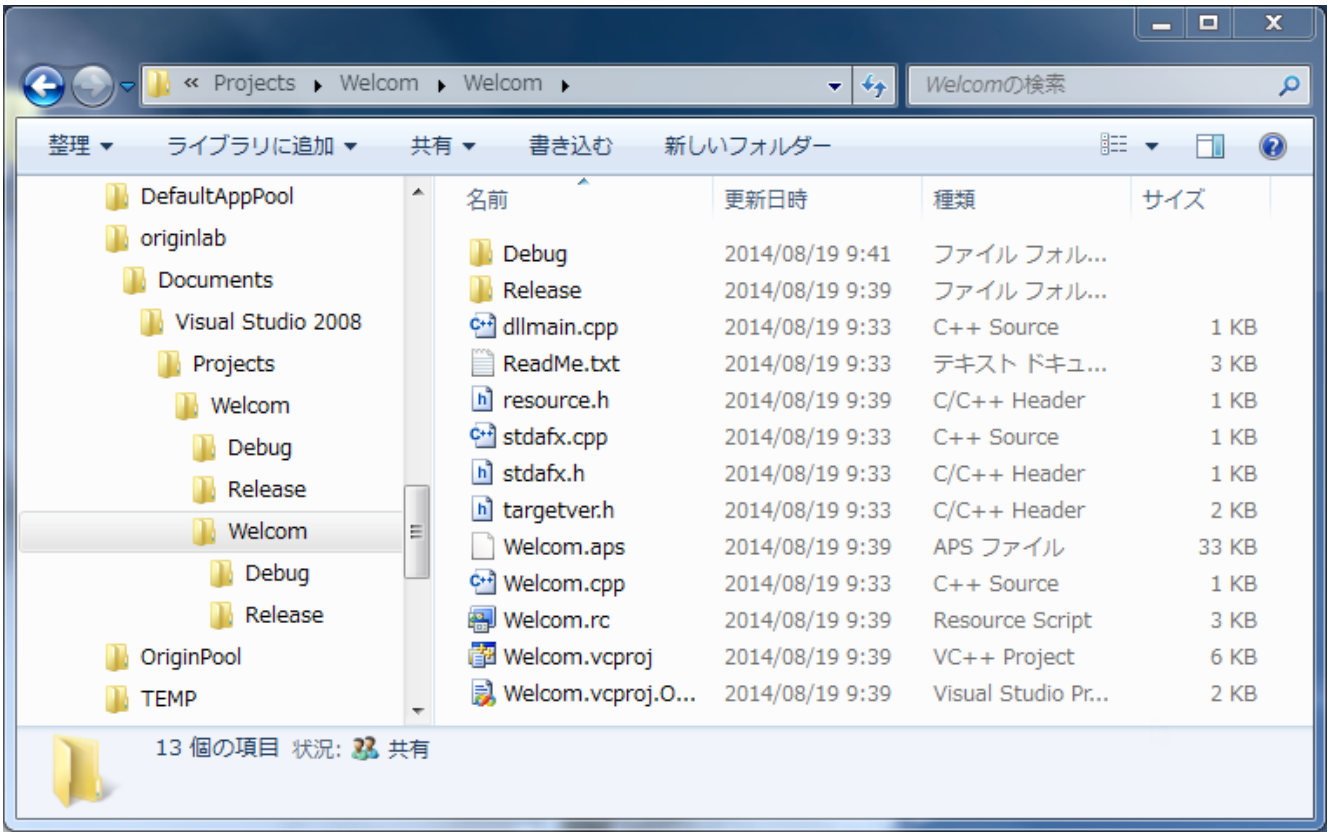
4. Win32 アプリケーションウィザードダイアログで、アプリケーションの種類を DLL にして、完了ボタンをクリックします。



5. プロジェクトのリソースビューに切り替え、プロジェクト名を右クリックして、リソースを追加し、リソースの種類は Dialog を選択して、新規作成をクリックします。

6. ソフトウェア環境に従って、リソースの言語プロパティをセットすることを忘れないようにしてください。英語のソフトウェアであれば、English(United States) です。

7. 希望のコントロールを追加し、プロジェクトの構成を Debug または Release にし、プロジェクトを保存します。そして、ビルド>ソリューションのビルドまたはソリューションのリビルドを選び、プロジェクトをビルドします。ソリューションフォルダに DLL を含む"Debug"または"Release"というフォルダが生成されます。このソリューションで生成したファイルは、次のものです。



Origin C でリソース DLL を使用する

このセクションでは、上記で作成したリソースのみの DLL を使用する方法を説明しています。

1. DLL ファイルを Debug または Release フォルダの外側にコピーし、DLL ファイルのパスを resource.h ファイルのパスと同じにします。
2. Origin を起動し、コードビルダを開きます。
3. DLL ファイルのパスに testODialog.c という新しい Origin C ファイルが作成されます。現在のワークスペースに追加し、次のようにテスト用のコードを記述します。OpenDlg 関数を実行し、ダイアログボックスを開きます。

```
#include <Dialog.h>

#include <..¥Originlab¥Resource.h> //ODialog リソースヘッダ

class MyDialog :public Dialog

{

public:

        // ダイアログ ID と DLL 名を持つダイアログを作成

        // "ODialog" は DLL ファイル名

        // パスを指定しなければ、DLL ファイルがこの Origin C ファイルと

        // 同じパスにあるということ

        // DLL が別の場所にある場合、DLL の

        // フルパスを使用

        MyDialog() :Dialog(IDD_OC_DIALOG, "ODialog")

        {

        }

};

void OpenDlg()

{

        MyDialog odlg;

        odlg.DoModal();

}
```

ウィザードダイアログ

このセクションは、Origin C でウィザードダイアログを開く方法を説明しています。このセクションのサンプルは、Origin C のデベロッパーキットと一緒にインストールされる既存のウィザードダイアログのリソース DLL を使用します。DLL は *Samples\DeveloperKit\Dialog Builder\Wizard* サブフォルダにあります。

ウィザードダイアログを開くには、最初にいくつかユーザ定義のクラスを定義します。**Dialog** クラスから派生するクラス、**WizardSheet** クラスから派生する別のクラス、**PropertyPage** クラスから派生する各ページのクラスが必要です。

WizardSheet::AddPathControl メソッドは、ウィザードのステップまたはページを移動する助けとなるウィザードマップを提供するのに使用します。マップをクリックして、ウィザード内のどのページにでもジャンプすることもできます。

定義した最初のクラスは、**PropertyPage** クラスから派生されています。この最初のクラスはウィザード内のすべてのページで共有されるすべての情報を含みます。

```
class WizPage :public PropertyPage

{

protected:

    WizardSheet* m_Sheet;

};
```

PropertyPage に基づくクラスを定義したので、ウィザードの各ページを取り扱うクラスを定義できます。これら次のクラスは上記で定義したページクラスから派生するものです。

```
class WizPage1 :public WizPage

{

};



class WizPage2 :public WizPage

{

};



class WizPage3 :public WizPage

{

};
```

定義される次のクラスは、プレースホルダクラスです。このクラスは、**WizardSheet** クラスから派生され、順に **PropertySheet** クラスから派生されます。このクラスは、データメンバーとしてすべてのページのインスタンスを保持します。

```
class WizSheet :public WizardSheet

{

public:

    // PropertySheet のデータメンバーは WizPage オブジェクト

    WizPage1 m_WizPage1;

    WizPage2 m_WizPage2;

    WizPage3 m_WizPage3;

};
```

全てのページとシートのクラスの定義が完了し、ダイアログクラスを定義できます。

```
class WizPageDialog :public Dialog

{

public:

    // メインダイアログのコンストラクタ

    WizPageDialog(int ID) :Dialog(ID, "Wizard.DLL")

    {

    }



    // メインダイアログのデータメンバーは PropertySheet（プレースフォルダ）

    WizSheet m_Sheet;

};
```

グラフプレビュー付きダイアログ

このセクションでは、グラフプレビュー付きカスタムダイアログの作成方法を示しています。

ダイアログリソースを準備

最初に、プレビューグラフが入れ子になった統計コントロールを含むダイアログリソースを作成します。ここでは、*OriginC\Originlab\ODlg8.dll* に組み込まれたリソースである、IDD_SAMPLE_SPLITTER_DLG を使用します。

ソースファイルを準備

コードビルダで、新規ボタン をクリックして、ファイル名を入力し、上述のダイアログリソースと同じパス（Origin のインストールフォルダの *OriginC\Originlab* サブフォルダ）にセットします。

必要なヘッダを含める

```
// これらのヘッダファイルは、ダイアログとコントロールの宣言を含む

#include <..¥Originlab¥DialogEx.h>

#include <..¥Originlab¥GraphPageControl.h>
```

ユーザ定義プレビュークラスを作成

```
// プレビューグラフのいくつかのアクションを禁止する

#define PREVIEW_NOCLICK_BITS (NOCLICK_DATA_PLOT|NOCLICK_LAYER|NOCLICK_LAYERICON)

#define PREVIEW_TEMPLATE    "Origin" // グラフテンプレートのプレビュー



class   MyPreviewCtrl

{

public:

    MyPreviewCtrl(){}

    ~MyPreviewCtrl()

    {

        // ダイアログを閉じたときに一時ブックを削除

        if ( m_wksPreview.IsValid() )

            m_wksPreview.Destroy();

    }



    void    Init(int nCtrlID, WndContainer& wndParent)

    {

        //プレビューグラフのコントロールを作成

        Control ctrl = wndParent.GetDlgItem(nCtrlID);

        GraphControl gCtrl;

        gCtrl.CreateControl(ctrl.GetSafeHwnd());

        gCtrl.Visible = true;
```

```
GraphPageControl gpCtrl;
gpCtrl.Create(gCtrl, PREVIEW_NOCLICK_BITS, PREVIEW_TEMPLATE);
GraphPage gpPreview;
gpPreview = gpCtrl.GetPage();
gpPreview.Rename("MyPreview");
m_glPreview = gpPreview.Layers(0); //第 1 レイヤ

if ( !m_wksPreview )
{
    //プレビューデータを持つ一時ワークシート
    m_wksPreview.Create("Origin", CREATE_TEMP);
    m_wksPreview.SetSize(-1, 2); //2 列

    //軸タイトルとしてロングネームを表示
    Column colX(m_wksPreview, 0);
    colX.SetLongName("Preview X");
    Column colY(m_wksPreview, 1);
    colY.SetLongName("Preview Y");

    //データ範囲を用意
    DataRange drPrev;
    drPrev.Add(m_wksPreview, 0, "X");
    drPrev.Add(m_wksPreview, 1, "Y");

    //プレビュー曲線をプロット。ここではポイントがない
    int nPlot = m_glPreview.AddPlot(drPrev, IDM_PLOT_LINE);
    DataPlot dp = m_glPreview.DataPlots(nPlot);
    if ( dp ) //プレビュー曲線色をセット
```

```
                dp.SetColor(SYSCOLOR_RED);
        }
    }

    //外部データとともにプレビュー曲線を更新
    void    Update(const vector& vX, const vector& vY)
    {
        if ( m_wksPreview.IsValid() )
        {
            Dataset dsX(m_wksPreview, 0);
            Dataset dsY(m_wksPreview, 1);
            if ( !dsX.IsValid() || !dsY.IsValid() )
                return; //プレビューのための列なし

            //ソースデータを更新するとプレビューグラフも更新
            dsX = vX;
            dsY = vY;
            //再スケール
            m_glPreview.Rescale();
        }
    }

private:
    //ダイアログ上のプレビューグラフ
    GraphLayer  m_glPreview;
    //プレビューデータを置くための一時ワークシート
    Worksheet   m_wksPreview;
};
```

ダイアログクラスを追加

```
class   MyGraphPreviewDlg :public MultiPaneDlg
{
public:
    //ダイアログリソース ID と それを含む DLL
    MyGraphPreviewDlg() :MultiPaneDlg(IDD_SAMPLE_SPLITTER_DLG,
        GetAppPath(TRUE) + "OriginC¥¥Originlab¥¥ODlg8")
    {
    }

    ~MyGraphPreviewDlg()
    {
    }

    int     DoModalEx(HWND hParent = NULL)
    {
        InitMsgMap();

        //ユーザが閉じるまでダイアログ表示
        return DoModal(hParent, DLG_NO_DEFAULT_REPOSITION);
    }

protected:
EVENTS_BEGIN
        ON_INIT(OnInitDialog)
        ON_BN_CLICKED(IDC_LOAD, OnDraw)
EVENTS_END
```

```
    //ダイアログイベントのメッセージハンドラー
    BOOL    OnInitDialog();
    BOOL    OnDraw(Control ctrl);

private:
    //プレビューコントロールを表すメンバー
    MyPreviewCtrl       m_Preview;
};

BOOL MyGraphPreviewDlg::OnInitDialog()
{
        m_Preview.Init(IDC_FB_BOX, *this);
        Button btn = GetItem(IDC_LOAD);
        if( btn )
                btn.Text = "Draw";
        return true;
}

BOOL MyGraphPreviewDlg::OnDraw(Control ctrl)
{
        vector vecX, vecY;
        vecX.Data(1.0, 10.0, 0.5);
        vecY.SetSize(vecX.GetSize());
        for(int ii = 0; ii < vecX.GetSize(); ++ii)
                vecY[ii] = rnd();
        m_Preview.Update(vecX, vecY);
        return true;
}
```

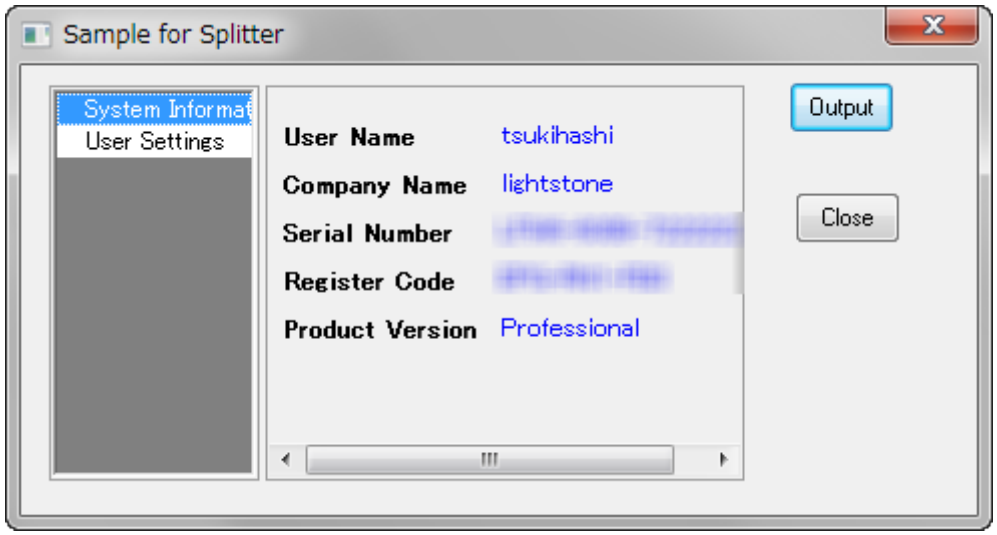
ダイアログを開く

```
void open_preview_dlg()

{

        MyGraphPreviewDlg dlg;

        dlg.DoModalEx(GetWindow());

        return;

}
```

上記関数を実行し、**Draw** ボタンをクリックします。すると、プレビューが更新されたのが確認できます。

Splitter ダイアログ

このサンプルは、ツリービューまたはグリッドビューを提供する Splitter ダイアログを作成する方法を示します。



ダイアログリソースを準備

このダイアログを作成するには、最初にラベルと 2 つのボタンコントロールを持つダイアログリソースを準備する必要があります。ここで、このサンプルを簡単にするため組込の *OriginC\Originlab\ODlg8.dll* ファイルにある既存のリソース IDD_SAMPLE_SPLITTER_DLG を使います。

ソースファイルを準備

コードビルダで、新規ボタン をクリックして、ファイル名を入力し、上述のダイアログリソースと同じパス（Origin のインストールフォルダの *OriginC\Originlab* サブフォルダ）にセットします。

ヘッダファイルをインクルードする

次のヘッダファイルがサンプルで使われます。上記で作成した ソースファイルに次をコピーします。

```
#include <..¥Originlab¥DialogEx.h>

#include <..¥Originlab¥SplitterControl.h>
```

```
#include <..¥Originlab¥DynaSplitter.h>
```

ユーザ定義 Splitter クラスを追加する

TreeDynaSplitter からクラスを派生することができます。ほとんどのダイアログの初期化と他のイベント関数のコードは、基底クラスにあり、splitter クラスを軽くします。Message Map

```
class MySplitter :public TreeDynaSplitter
{
public:
        MySplitter(){}
        ~MySplitter(){}
        //splitter コントロールを初期化
        int         Init(int nCtrlID, WndContainer& wndParent, LPCSTR lpcszDlgName = NULL)
        {
                TreeDynaSplitter::Init(nCtrlID, wndParent, 0, lpcszDlgName);
                return 0;
        }
        //現在の文字列を出力
        void        Output()
        {
                out_tree(m_trSettings);
        }
protected:
        // メッセージマップテーブルとメッセージハンドラを宣言
        DECLARE_MESSAGE_MAP
        BOOL        OnInitSplitter();
        BOOL        InitSettings();
        void        OnRowChange(Control ctrl);

private:
```

```
        BOOL        constructSettings();
        BOOL        initSystemInfo(TreeNode& trSys);// システム情報を表示
        BOOL        initUserInfo(TreeNode& trUser);//ユーザ設定を収集

private:
        GridTreeControl m_List; //左パネルのグリッド表示
        Tree    m_trSettings;//右パネルのグリッド表示
        bool        m_bIsInit;//初期化イベントからかどうかを指示

};

//コントロールメッセージとイベントをマッピング
BEGIN_MESSAGE_MAP_DERIV(MySplitter, TreeDynaSplitter)

        ON_INIT(OnInitSplitter) //splitter 設定初期化
        //破棄時に splitter のサイズと位置を保存
        //これは基底クラスで行われる
        ON_DESTROY(OnDestroy)
        ON_SIZE(OnCtrlResize)
        //コントロールが準備できたら、splitter と位置をリサイズ
        ON_USER_MSG(WM_USER_RESIZE_CONTROLS, OnInitPaneSizs)

        //ユーザは左パネルの異なる行を選択
        ON_GRID_ROW_COL_CHANGE(GetMainPaneID(), OnRowChange)
END_MESSAGE_MAP_DERIV

BOOL        MySplitter::OnInitSplitter()
{
```

```
        TreeDynaSplitter::OnInitSplitter(&m_List);
        constructSettings();  //ツリー設定を構築
        InitSettings(); //ツリー設定を splitter GUI に
        SetReady();
        return TRUE;
}

//ユーザが異なる行を選択するとき、右パネルを更新
void       MySplitter::OnRowChange(Control ctrl)
{
        if ( !m_bReady )
                return;
        //現在のブランチのサブノードを表示
        TreeNode trCurrent = ShowListContent(-1, true, m_bIsInit);
        if ( trCurrent )
        {
                //レジストリから設定をロード
                string strTag = trCurrent.tagName;
                LoadBranchSetting(GetDlgName(), strTag);
        }
        m_bIsInit = false;
        return;
}

//splitter 設定を初期化
BOOL       MySplitter::InitSettings()
{
        m_bIsInit = true;
```

```
        ///GUI の問題を避けるため、準備状態がセットしない
        m_bReady = false;
        //表示のための splitter tree をセット
        ShowList(m_trSettings, ATRN_STOP_LEVEL);
        m_bReady = true; //準備状態をリセット
        SelectRow(0); //最初の行を選択
        return TRUE;
}

BOOL       MySplitter::constructSettings()
{
        TreeNode trSys = m_trSettings.AddNode("System");
        trSys.SetAttribute(STR_LABEL_ATTRIB, "System Information");
        initSystemInfo(trSys);

        TreeNode trUser = m_trSettings.AddNode("User");
        trUser.SetAttribute(STR_LABEL_ATTRIB, "User Settings");
        initUserInfo(trUser);
        return TRUE;
}

//Origin の基本情報を表示
//OS 関連の情報も表示可能
BOOL       MySplitter::initSystemInfo(TreeNode& trSys)
{
        if ( !trSys )
                return FALSE;
```

```
char szUser[LIC_USERINFO_NAME_COMPANY_MAXLEN];
char szCompany[LIC_USERINFO_NAME_COMPANY_MAXLEN];
char szSerial[LIC_OTHER_INFO_MAXLEN];
char szRegCode[LIC_OTHER_INFO_MAXLEN];
DWORD dwProd = GetLicenseInfo(szUser, szCompany, szSerial, szRegCode);
string strProduct;
switch( dwProd & 0x000000FF )
{
case ORGPRODUCTTYPE_EVALUATION:
        strProduct = "Evaluation";
        break;
case ORGPRODUCTTYPE_STUDENT:
        strProduct = "Student";
        break;
case ORGPRODUCTTYPE_REGULAR:
        strProduct = "Regular";
        break;
case ORGPRODUCTTYPE_PRO:
        strProduct = "Professional";
        break;
default:
        strProduct = "Unknown";
        break;
}

GETN_USE(trSys)
GETN_STR(UserName, "User Name", szUser)
GETN_READ_ONLY_EX(2)
```

```
        GETN_STR(Company, "Company Name", szCompany)
        GETN_READ_ONLY_EX(2)
        GETN_STR(SeriNum, "Serial Number", szSerial)
        GETN_READ_ONLY_EX(2)
        GETN_STR(RegCode, "Register Code", szRegCode)
        GETN_READ_ONLY_EX(2)
        GETN_STR(Product, "Product Version", strProduct)
        GETN_READ_ONLY_EX(2)
        return TRUE;
}

//ユーザ情報と設定を収集するコントロール
BOOL       MySplitter::initUserInfo(TreeNode& trUser)
{
        if ( !trUser )
                return FALSE;

        GETN_USE(trUser)
        GETN_STRLIST(Language, "Language", "English", "|English|German")
        GETN_STR(UserID, "User ID", "")
        GETN_PASSWORD(Password, "Password", "")
        GETN_STR(Email, "Email", "user@originlab.com")

        return TRUE;
}
```

ユーザ定義 Splitter ダイアログクラスを追加する

splitter ダイアログは、splitter コントロールオブジェクトを含み、ダイアログは splitter コントロールを初期化し、適切なイベントにメッセージを渡します。

```
//ダイアログ名は、レジストリに設定を保存するのに使用

#define    STR_DLG_NAME    "My Splitter Dialog"

class MySplitterDlg : public MultiPaneDlg

{

public:

        //リソース ID とその DLL はダイアログリソースを含む

        MySplitterDlg() : MultiPaneDlg(IDD_SAMPLE_SPLITTER_DLG, "ODlg8")

        {

        }

        ~MySplitterDlg()

        {

        }

        //ユーザが閉じるまでダイアログを開く

        int         DoModalEx(HWND hParent = NULL)

        {

                //メッセージマップをセット

                InitMsgMap();

                return DoModal(hParent, DLG_NO_DEFAULT_REPOSITION);

        }

        //ダイアログを開く前にコントロールと他の設定を初期化

        BOOL        OnInitDialog();

        //ダイアログ初期化が完了したとき

        BOOL        OnReady();

        //ユーザが'Output'ボタンをクリックしたとき

        BOOL        OnOutput(Control ctrl);

protected:

        DECLARE_MESSAGE_MAP

private:
```

```
        MySplitter      m_Splitter;


};


//マップダイアログメッセージ
BEGIN_MESSAGE_MAP(MySplitterDlg)
        ON_INIT(OnInitDialog)
        ON_READY(OnReady)
        ON_BN_CLICKED(IDC_LOAD, OnOutput)
END_MESSAGE_MAP


BOOL        MySplitterDlg::OnInitDialog()
{
        //ボタンのテキストを意味のあるテキストに変更
        GetDlgItem(IDC_LOAD).Text = "Output";
        GetDlgItem(IDCANCEL).Text = "Close";
        m_Splitter.Init(IDC_FB_BOX, *this, STR_DLG_NAME);
        return TRUE;
}


BOOL        MySplitterDlg::OnReady()
{
        //ダイアログを更新
        UpdateDlgShow();
        SetInitReady();
        //準備状態を位置とサイズを初期化
        m_Splitter.OnReady();
        return TRUE;
```

```
}



BOOL        MySplitterDlg::OnOutput(Control ctrl)

{

        //現在のユーザ設定を出力

        m_Splitter.Output();

        return TRUE;

}
```

ダイアログを開く

上記のステップの後、すべてのコードを保存し、ビルドし、次の関数を実行して、スプリッターダイアログを実行します。

```
void        test_MySplitterDlg()

{

        MySplitterDlg dlg;

        dlg.DoModalEx(GetWindow());

}
```

18.1.6 JavaScript サポート付 Origin C HTML ダイアログ

JavaScript サポート付 Origin C HTML ダイアログ

HTML ダイアログ Origin C では、DLL を使わずに、HTML で Origin 内にダイアログボックスを構築できます。Visual Studio を使って DLL リソースを構築する必要がなくなっただけではなく、ウェブページ作成用のパブリックドメインを幅広く利用することを可能にします。HTML ダイアログの要素にアクセスしてコントロールするため、Origin2017 に JavaScript インテグレーションが加えらました。JavaScript を呼び出す OriginC の手法、および OriginC を呼び出す JavaScript の手法が導入されています。また、Origin のグラフコントロールが HTML コントロールを重ねることができるので、この 2 つをダイアログ内で調整し、きれいに配置することができます。

このチュートリアルでは、HTML ページを作成する方法と、OriginC を使ってダイアログ内のコントロールとして HTML ページを表示する方法について解説します。さらに、グラフコントロール付 HTML の作成する方法と、JavaScript の内容を呼び出して OriginC を実行(また、その逆を実行)する方法についても説明しています。

サンプル

チュートリアルで使う、cpp ファイル、html ファイル、関連 css ファイル、js ファイルを含む HTML ダイアログサンプルファイルは、こちらからダウンロード可能です。

以下のチュートリアルに含まれるファイルの他、
\HTMLDlgExamples\OC HTML Dialog\Other Examples サブフォルダーに追加のサンプルがあります。これらのサンプルは様々な分野をカバーしています。

単純な Hello World ダイアログ Hello World ダイアログ HTML ダイアログ

サマリー

このチュートリアルでは、以下の項目について説明します。

1. HTML ページの作成
2. OriginC を使ってダイアログボックス内で HTML ページを表示
3. ダイアログのイベントに対応するイベントハンドラーのビルド

必要な Origin のバージョン: Origin 2017 以降

サンプルファイル

- index.html: html ページの HTML コード
- HelloWorldDlg.cpp: HTML ページを呼び出してダイアログ内でそれを表示する OriginC のコード
- HelloWorldDlg1.cpp: 応用的な機能を含む OriginC のコード

Note: こちらから、サンプルファイルをダウンロードできます。

HTML ページの作成

HTML ダイアログを作成する際には、まず初めに HTML ページを作成します。

1. Origin を起動して、 をダブルクリックして**コードビルダ**を開きます。
2. 「index.html」という新しい HTML ファイルを作成して、「Hello World」という新しいフォルダに保存します。
3. < head >と< body >要素を中に入れ、< title >, < h1 >, < p > などの別の HTML 要素を加えて HTML ページを構築します。

```
<!DOCTYPE html><html>     <meta charset="utf-8" />     <meta http-equiv="X-UA-Compatible"
content="IE=Edge"/>     <head>        <title>           A Small
Hello        </title>     </head>     <body>         <h1>Hello World</h1>         <p>This is very
minimal "hello world" HTML document.</p>     </body></html>
```

Note:編集を終えたら、ブラウザでこのページを開いて、確認します。

HTML ダイアログの作成

このセクションでは、ダイアログの HTML ページ表示を作成します。

1. **コードビルダ**で、HTML ファイルのパスの下に、新しい cpp ファイル「DlgWithGraph.cpp」を作成します。

2. まず初めに、3 つの必要なヘッダを入れます。ここで、HTMLDlg.は Origin 2017 で新しくなっており、Origin と HTML ダイアログを一緒に結びつけます。

```
#include <Origin.h> #include <..¥OriginLab¥DialogEx.h> #include <..¥OriginLab¥HTMLDlg.h>
```

3. ユーザ定義 HTML ダイアログクラスを HTMLDlg クラスから生成し、手法 GetInitURL()の中に HTML ページを示します。

```
class HelloWorldDlg: public HTMLDlg {     public:          string GetInitURL() //html ファイルのパスを得ます。                    {              string strFile = __FILE__; //現在のファイルのパス     return GetFilePath(strFile) + "index.html"; } };
```

4. HTML ダイアログを開くには、すべてのコードを保存してビルドし、以下の関数を実行します。

```
void hello() {     HelloWorldDlg dlg;     dlg.DoModalEx(GetWindow()); }
```

「Hello World」の付いた HTML ダイアログがポップアップされます。

高度な機能

ダイアログを変更してイベントヘッダーを追加し、プログラムを改良することができます。

> **Note:** 以下の機能の完全なコードは「HelloWorldDlg1.cpp」で利用可能です。

ダイアログのサイズ変更を禁止

この設定を行うには、ModifyStyle を使い、ダイアログクラスの手法 OnInitDialog「WS_THICKFRAME」を無効にします。

```
BOOL OnInitDialog() {     ModifyStyle(WS_THICKFRAME, 0);//サイズ変更を禁止     return HTMLDlg::OnInitDialog(); }
```

ダイアログサイズの設定

対のクラス手法 GetDlgInitSize をダイアログクラスから呼び出し、ダイアログの幅と高さを指定する必要があります。

```
BOOL GetDlgInitSize(int& width, int& height)//ダイアログサイズを得ます。 {     width = 500;     height =
200;     return true; }
```

イベントハンドラーの追加

ダイアログビルダと同様に、メッセージマップを使って、どのイベントを扱うのか、どの関数を呼び出すのかを指定します。
例えば、イベントハンドラー手法をユーザ定義のクラス「HelloWorldDlg」に 加えて、ダイアログを閉じるとメッセージがポップアップするようにします。

1. クラス HelloWorldDlg にある以下のメッセージマップにより、ダイアログメッセージをマップします。

```
//メッセージマップ
public:     EVENTS_BEGIN_DERIV(HTMLDlg)          ON_DESTROY(OnDestroy)      EVENTS_END_DERIV
```

2. イベントハンドラー手法をユーザ定義のクラス HelloWorldDlg に加えて、ダイアログを閉じるとメッセージがポップアップするようにします。

```
BOOL OnDestroy() {     MessageBox(GetSafeHwnd(), _L("Thank you and have a good
day!"),_L("Message"));     return true; }
```

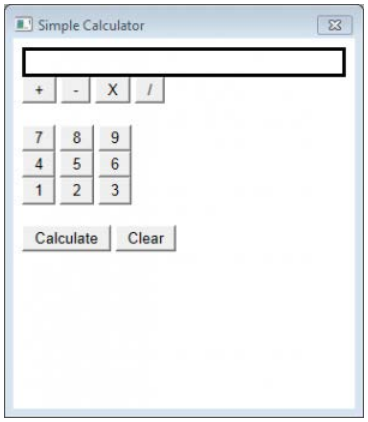
計算機の作成 CalculatorHTML ダイアログ

サマリー

このチュートリアルでは、算術演算（足し算、引き算、掛け算、割り算）を実行する計算機を構築します。
ここでは、以下の項目を解説します。

1. HTML ページ内のコントロールを使って発生するトリガーイベント
2. イベントハンドラー関数を構築して JavaScript によるイベントに応答します。
3. OriginC 関数を JavaScript で呼び出します。
4. OriginC で基本的な算術演算を実行します。

これにより、以下の画像のような計算機になります。



必要な Origin のバージョン: Origin 2017 以降

サンプルファイル

- SimpleCalc.html: html ページの HTML コード
- Calculator.cpp: HTM ダイアログを呼び出して算術演算を実行する OriginC のコード

Note: こちらから、サンプルファイルをダウンロードできます。

計算機用の HTML ページの作成

この計算機を作には、まず初めに、HTML ページを作成して、算術式表示ボックス、数字入力ボタン、オペレーターボタン、計算ボタン、消去ボタンをこのページに追加します。

1. Origin を起動して、**コードビルダ**を開き、SimpleCalc.html という名前の新しい HTML ファイルを作成します。Simple Calculator という名前の新しいフォルダに、このファイルを保存します。
2. SimpleCalc.html の中に、以下のコードを使って、計算機用のシンプルなインターフェースを作成します。

```
<!DOCTYPE html><html>       <head>         <meta charset="utf-8" /> <meta http-equiv="X-UA-
Compatible" content="IE=Edge"/>     </head>       <body>         <!--equation display textbox--> <div
id="Equation" style="border:solid;height:20px;"></div>         </br>           <!--four arithmetic
operations buttons--> <div id="Ops">           <input type="button" id="btnPlus" value="+"/>
9-->         <div id="Num789">               <input type="button" id="btn7" value="7"/>
value="5"/>               <input type="button" id="btn6" value="6"/>         </div>         <div
id="Num123">              <input type="button" id="btn1" value="1"/>                   <input
type="button" id="btn2" value="2"/>           <input type="button" id="btn3"
value="3"/>         </div> <div id="Num0"> <input type="button" id="btn0" value="0"/>
</div>         </br>   <!--calculation button and clear button     -->         <div>
      <input type="button" id="btnCalculate" value="Calculate"/>            <input type="button"
id="btnClear" value="Clear"/>         </div>     </body>  </html>
```

Note:編集を終えたら、ブラウザでこのページを開いて、確認します。

ボタン用のワンクリックヘッダーを追加

以下のように計算機は動作します。

1. 数字入力ボタンをクリックすると、プログラムによってテキストボックスに入力数字が表示されます。
2. オペレーターボタンをクリックすると、先にインプットした数字にオペレーターが追加されます。
3. テキストボックスにも追加する別の数字をクリックして入力します。
4. 最後に、計算ボタンをクリックしてテキストボックスの算術式と結果を確認します。
5. テキストボックスのテキストは消去ボタンをクリックして、いつでも消去することができます。

この方法でプログラムを実行するには、onclick イベントハンドラーを追加して、HTML ページのボタンをマウスでクリックすることを追跡する必要があります。この操作により、計算を実行する際に使う JavaScript が呼び出されます。

計算ボタン Calculate を使う:

1. 要素< script >の中に、関数 Invoke_Cal()を加えます。ここで、この関数を使って、手順 2 に記載してある onclick イベントハンドラで、ボタンのクリックに応答するようにします。

```
<script>        function Invoke_Cal() {}      </script>
```

2. 計算ボタンタグの中にワンクリックハンドラーイベントを追加して、ボタンクリックによって関数 Invoke_Cal()が起動されます。

```
<input type="button" id="btnCalculate" value="Calculate" onclick="Invoke_Cal()"/>
```

3. これまでの手順と同様に、onclick イベントハンドラをボタンすべてに加えてクリック操作を追跡して、<script>内のの対応する JavaScript 関数に追加することができます。

OriginC 関数を JavaScript で呼び出します。

このサンプルでは、OriginC で算術演算を実行してみることにより、JavaScript の OriginC を呼び出す方法を説明します。JavaScript で関数 window.external.ExtCall を使って、OriginC を使います。

```
window.external.ExtCall( “OriginC Function Name”, Parameter1, Parameter2…)
```

Note:

1. 5 つ以下のパラメータが OriginC 関数名を含んで ExtCall に渡すことができる場合、
2. OriginC の入力引数として渡されるパラメータは、bool, integer, double , string を含むプリミティブ型のみになります。

JavaScript の OriginC を呼び出す方法を理解した後、以下のように<script>内の関数を変更することができ、HTML サンプルファイルでコンプリートコードを利用できます。

```
<script>    var PlusOp = "+";    var MinusOp = "-";    var MultiplyOp = "*";    var DivOp =
"/";      function Invoke_AddOp(strOp) {    var OriginStr =
document.getElementById(&apos;Equation&apos;).innerHTML;        //OriginC 関数 AddOp を呼び出して、テキストボックスにオペレータシンボルを表示します。        var NewStr = window.external.ExtCall("AddOp", OriginStr,
strOp);        document.getElementById(&apos;Equation&apos;).innerHTML = NewStr;    }      function
Invoke_AddNum(NewNum) {        var OriginStr =
document.getElementById(&apos;Equation&apos;).innerHTML;        //OriginC 関数 AddNum を呼び出して、テキストボックスに入力数字を表示します。        var NewStr = window.external.ExtCall("AddNum", OriginStr, NewNum);
    document.getElementById(&apos;Equation&apos;).innerHTML = NewStr;    }      function Invoke_Cal() {        va
Str);        document.getElementById(&apos;Equation&apos;).innerHTML = Result;    }      function
Invoke_Clear() {        //OriginC 関数 Clear を呼び出してテキストボックスのテキストを消去します。
    document.getElementById(&apos;Equation&apos;).innerHTML = window.external.ExtCall("Clear");    }</script>
```

HTML ダイアログの作成

このセクションでは、OriginC のコードを編集して実行し、Origin に計算機ダイアログを作成します。

1. **コードビルダ**で、SimpleCalc.html のパスの下に、新しい cpp ファイル Calculator.cpp を作成します。
2. 必要なヘッダの追加します。

```
#include <Origin.h> #include <..¥OriginLab¥DialogEx.h> #include <..¥OriginLab¥HTMLDlg.h>
```

3. ユーザ定義 HTML ダイアログクラスをクラス HTMLDlg から生成します。

```
class OriginCalculatorDlg: public HTMLDlg {     public:         string
GetInitURL()             {                 return GetFilePath(__FILE__) +
"SimpleCalc.html";             } string GetDialogTitle()             {                 return "Simple
Calculator";             } };
```

4. 計算機ダイアログを作成するメイン関数 calc を追加します。

```
void calc() {     OriginCalculatorDlg dlg;     dlg.DoModalEx(GetWindow()); }
```

5. 全てのコードを保存してビルドし、calc() 関数を実行してダイアログを開きます。

OriginC でプロトタイプを呼び出し可能な JavaScript を追加

このセクションでは、JavaScript で呼び出すダイアログクラス OriginCalculatorDlg を追加して基本的な算術演算を実行します。

1. ダイアログクラスの Add DECLARE_DISPATCH_MAP を追加して、JavaScript で関数を配置します。

```
public:     DECLARE_DISPATCH_MAP
```

2. 算術演算を実行する手法を追加して、テキストボックス内の式を表示・削除します。

```
//オペレータのどれかをクリックする場合 string OriginCalculatorDlg::AddOp(string str, string strOp)
{     return str + " " + strOp + " "; } //数字のどれかをクリックする場合 string
OriginCalculatorDlg::AddNum(string str, int NewNum) {     return str + ftoa(NewNum); }  // LabTalk 表
記を使って計算 double OriginCalculatorDlg::Calculate(string str) {     double
dd;     LT_evaluate(str ,&dd);     return dd; }   string OriginCalculatorDlg::Clear(void)
{     return ""; }
```

3. ダイアログクラスの手法を宣言

```
public:     string AddOp(string str, string strOp);     string AddNum(string str, int
NewNum);     double Calculate(string str);     string Clear();
```

4. OriginC クラスの手法を HTML ダイアログにマップします。

```
BEGIN_DISPATCH_MAP(OriginCalculatorDlg, HTMLDlg)     DISP_FUNCTION(OriginCalculatorDlg, AddOp,
VTS_STR, VTS_STR VTS_STR)     DISP_FUNCTION(OriginCalculatorDlg, AddNum, VTS_STR, VTS_STR
VTS_I4)     DISP_FUNCTION(OriginCalculatorDlg, Calculate, VTS_R8,
VTS_STR)     DISP_FUNCTION(OriginCalculatorDlg, Clear, VTS_STR, VTS_VOID) END_DISPATCH_MAP
```

Note: DISP_FUNCTION のシンタックスは、以下のようになります。

```
DISP_FUNCTION(-defined Dialog Class, Function Name, Type of Ouput, Type of Input Type of
Input…)
```

ここで、VTS_BOOL = bool, VTS_I4 = integer, VTS_R8 = double, VTS_STR = string and VTS_VOID = void です。

6. コードを保存してビルドし、calc()関数を再度実行すれば、この簡易計算機で計算ができます。

グラフ付き HTML ダイアロググラフ付 HTML ダイアログ HTML ダイアログ

サマリー

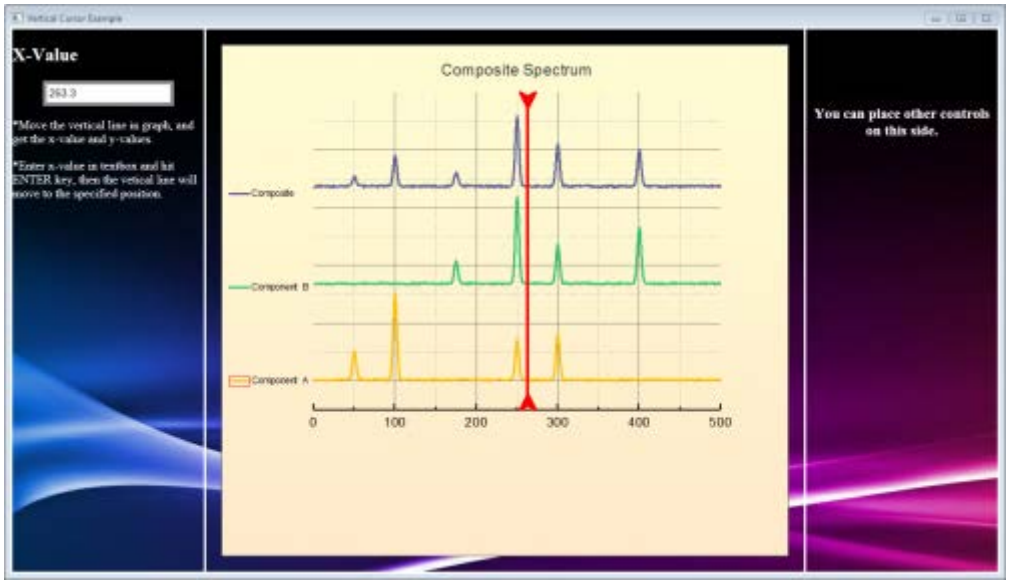
このチュートリアルでは以下のようにダイアログ機能を構築します。

1. ダイアログには 3 つのパネルがあります。このうち、左のパネルにテキストボックス、中央パネルにグラフがあり、右のパネルはブランクになっています。
2. 手動でグラフに垂直線を移動すると、プログラムによってテキストボックスに X の値が表示されます。
3. また、テキストボックスの X の値を入力して Enter キーを押すと、プログラムによって垂直線は指定した位置に移動します。

ここでは、以下の項目を解説します。

1. HTML ダイアログでグラフを含む方法
2. 好きなようにダイアログのサイズを変更する方法
3. JavaScript で OriginC を呼び出す方法（チュートリアル計算機の作成にも解説があります。
4. Origin C で JavaScript を呼び出す

このチュートリアルを終えると、異なる番号を除いて以下の画像のようにダイアログが表示されます。



必要な Origin のバージョン: Origin 2017 以降

サンプルファイル

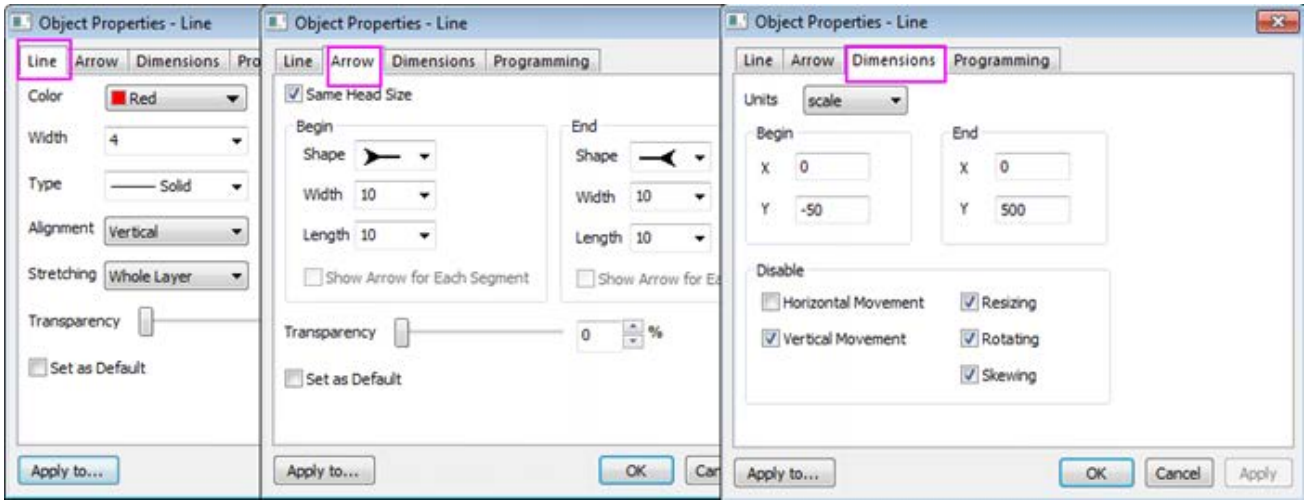
1. Composite Spectrum.opj: この OPJ にはダイアログに表示されるグラフを含んでいます。
2. index.html: html ページの HTML コード
3. DlgWithGraph.cpp: HTML ダイアログを呼び出して、OriginC と JacaScript の連携を行う OriginC のコードです。
4. Background_image1.png: ダイアログの背景用の最初の画像です。
5. Background_image2.jpg: ダイアログの背景用の二番目の画像です。

Note: こちらから、サンプルファイルをダウンロードできます。

準備

1. 「Composite Spectrum.opj」という Origin のプロジェクトを作成して、「HTML Dialog with A Graph」フォルダに保存します。
2. Origin のメニューから**ファイル: インポート: 単一 ASCII** を選択して「\Samples\Curve Fitting」にあるデータ「Composite Spectrum.dat」をインポートします。
3. **列 B** から**列 D** を選択して、メニューから**作図＞2D: 複数 Y 軸軸: Y オフセット付き積み上げ折れ線**を選択して折れ線グラフを作成します。
4. **Graph1** をダブルクリックして**作図の詳細**ダイアログを開き、**Graph 1** のフォーマットを変更します。
5. **直線ツール**ボタン をクリックしてグラフに垂直線を追加します。(SHIFT キーを押して描画します。
6. 垂直線を右クリックして、コンテキストメニューから**プロパティ**を選択します。

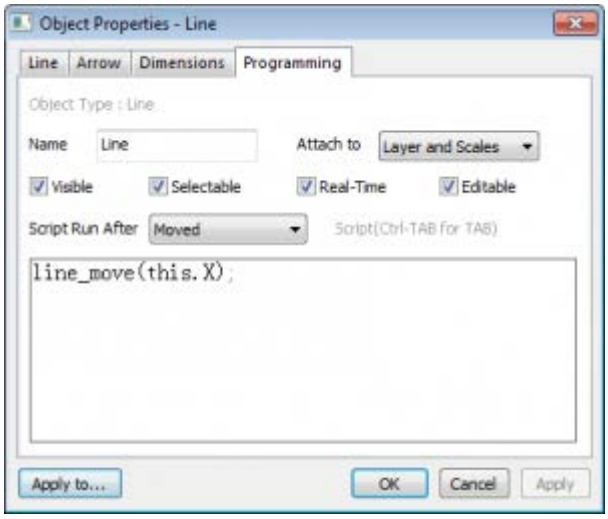
7. **線**のタブと**矢印**のタブを開いて、垂直線の形式を変更し、**サイズ**タブで水平移動のみが有効になるように設定します。



8. **プログラミング**のタブでオブジェクトに **Line** という名前を付けます。

9. **のあとでスクリプトを実行**のドロップダウンリストから**移動**を選択して、テキストボックスに以下の LabTalk スクリプトを入力します。

```
line_move(this.X);
```

10. 

> **Note:** 垂直線を移動する場合、この手順で関数 line_move()のトリガを設定します。この関数 line_move()は、後程、OriginC で作成します。

11. **コードビルダ**を開き、**ワークスペース**ウィンドウの**プロジェクト**フォルダを開きます。「ProjectEvents.OGS」をダブルクリックしてファイルを開き、[AfterOpenDoc]のセクションにスクリプトを追加します。

```
if(run.LoadOC("%X¥DlgWithGraph.cpp", 16) == 0) {     @G=0;//両サイドに灰色のバンドが無くなるようにグラフの背景を塗りつぶし     HTMLandGraphDlgEx; //ダイアログを立ち上げる関数 } else {     type "Failed to load the dialog!";     return 0; }
```

> **Note:** Origin のプロジェクトを開いた後、このスクリプトによりダイアログが起動します。ProjectEvents スクリプトについての詳細はこちらをご覧ください。

13. **コードビルダ**の ProjectEvents.OGS を保存して、Origin にプロジェクトを保存します。

ダイアログ用の HTML ページの作成

HTML ダイアログを作成する際には、まず初めに HTML ページを作成します。

1. 「HTML Dialog with A Graph」フォルダに背景用の画像を 2 つ用意します。
2. **コードビルダ**を開き、新しい HTML ファイルを作成して、「index.html」という名前で「HTML Dialog with A Graph」フォルダに保存します。
3. 以下のコードをコピーして「index.html」内に貼り付けます。

```
<!DOCTYPE html><html>    <head> <meta charset="utf-8" />        <meta http-equiv="X-UA-Compatible"
content="IE=Edge" />    <style>    body{        background-
image:url("background_image1.png")    }    input[type="text"] {        font-size:
14px;    }    input[type="text"]:focus {        outline: none;    }    .style-2 input[type="text"]
{        padding: 3px;        border: solid 5px #c9c9c9;        transition: border
0.3s;    }    .style-2 input[type="text"]:focus,    .style-2 input[type="text"].focus
{        border: solid 5px #969696;    }    #container {        border: 3px solid
#ffffff;        position: absolute;        left: 0;        right: 0;        bottom: 0;        top:
0;    }    #leftcontainer {        border-right: 3px solid #ffffff; width: 250px; height: 100%;
float: left;    }    #rightcontainer { border-left: 3px solid #ffffff; width: 250px; height: 100%;
float: right; text-align: center;    } </style>    </head>    <body> <div id="container">    <div
id="leftcontainer">                <h2><font color="ffffff">X-Value</font></h2> <ul class="input-
list style-2 clearfix">    <input type="text" name="XValue"  id="X" placeholder="X"  onkeydown = "">
</ul>   <p><font color="ffffff">*Move the vertical line in graph, and get the x-value and y-
values.</font></p> <p><font color="ffffff">*Enter x-value in textbox and hit ENTER key, then the
vetical line will move to the specified position.</font></p>    </div>    <div id="rightcontainer">
<h3>   </h3> <h3>   </h3> <h3><font color="ffffff">You can place other controls on this
side.</font></h3>    </div>        </div>    </body></html>
```

この手順を終了すると、ウェブブラウザで「index.html」が開き、以下のページが表示されます。

ダイアログ用の JavaScript コードの追加

この手順で、JavaScript コードを追加します。

1. 以下の JavaScript コードをコピーして html ファイルの<script>内に貼りつけます。

```
<script>    var vbg = 1;//html 背景を交換する    function lineMove(xVal)    {        var XValue =
document.getElementById("X"); XValue.value = xVal; //html 背景を変更を表示し、グラフには影響しない。
switch (vbg) {    case 1:        document.body.style.backgroundImage =
"URL(&apos;background_image2.jpg&apos;)";        vbg = 2;    break;    case 2:
document.body.style.backgroundImage = "URL(&apos;background_image1.png&apos;)"; vbg =
1;    break; }    }        function enterXval()    { var XValue =
document.getElementById("X");        window.external.ExtCall("OnEnterXvalToUpdateGraph",
XValue.value);    }        function getGraphRect()    { var leftDiv =
document.getElementById("leftcontainer"); var leftpos = leftDiv.getBoundingClientRect().right; var
toppos = leftDiv.getBoundingClientRect().top; var bottompos =
leftDiv.getBoundingClientRect().bottom;   var rightDiv = document.getElementById("rightcontainer");
var rightpos = rightDiv.getBoundingClientRect().left;   return JSON.stringify({    left: leftpos +
20,    top: toppos + 20,    right: rightpos - 20,    bottom: bottompos - 20})    }</script>
```

Note:このコードで、機能を HTML ページに追加します。

- lineMove():垂直線を移動する場合に、テキストボックスの X 値を更新します。
- enterXval(): テキストボックスの Enter キーを押すことに対応します。
- getGraphRect(): グラフが表示されている四角形のボックスを返します。

2. Enter を押すことをキャプチャするため、onkeydown HTML コードのテキストボックスタブにあるイベントハンドラーを追加します。これは、Enter キーを押して関数 enterXval()を起動するためです。

```
<input type="text" name="XValue"  id="X" placeholder="X"  onkeydown = "javascript:if(event.keyCode
== 13) enterXval();">
```

1. 関数 lineMove()では, switch 文の追加を推奨しています。コメントの通り、グラフが HTML ページに重なり、背景を変更しても影響しないことが分かります。
2. 関数 enterXval(), window.external.ExtCall は Origin C を呼び出す時に使います。window.external.ExtCall についての詳細は、こちらをご覧ください。

グラフ付 HTML ダイアログの作成

OriginC コードを編集して HTML ダイアログを作成する準備ができました。

1. **コードビルダ**で、「HTML Dialog with A Graph」フォルダの下に、新しい cpp ファイル「DlgWithGraph.cpp」を作成します。

2. 必要なヘッダの追加します。

```
#include <Origin.h> #include <../OriginLab/DialogEx.h> #include <../OriginLab/HTMLDlg.h> #include
<../OriginLab/GraphPageControl.h>
```

3. GRAPH_CONTROL_ID を定義します。

```
#define GRAPH_CONTROL_ID 1
```

4. ユーザ定義 HTML ダイアログクラス HTMLandGraphDlg を HTMLDlg クラスから生成し、手法 GetInitURL()の中に HTML ページを示します。

```
class HTMLandGraphDlg: public HTMLDlg {     protected:          string GetInitURL() //html ファイルのパスを得る                    {     string strFile = __FILE__; //現在のファイル名     return
GetFilePath(strFile) + "index.html"; } string GetDialogTitle() {return "Vertical Cursor Example";} //ダイアログ名の設定 };
```

5. イベントハンドラー手法を HTMLandGraphDlg に追加して、ユーザがダイアログ開く、サイズ変更、閉じるトリガーとなるイベントに応答します。

```
private:     GraphControl     m_gcCntrl; protected:     BOOL OnInitDialog() //ダイアログが初めて表示された場合は、ダイアログを初期化する必要があります。      {          LT_execute(";doc -m 0");//Origin メインウィンドウを隠す HTMLDlg::OnInitDialog(); //HTMLDlg クラスからダイアログを生成 ModifyStyle(0,
WS_MAXIMIZEBOX);//標準的な HTML ダイアログリソースには、最大化ボタンがありません。 RECT rr;
m_gcCntrl.CreateControl(GetSafeHwnd(), &rr, GRAPH_CONTROL_ID, WS_CHILD|WS_VISIBLE|WS_BORDER);   //グラフの様々な構成要素をクリックすることを無効にするオプションを設定します。「OC_const.h」に詳細があります。 DWORD dwNoClicks = NOCLICK_AXES | NOCLICK_DATA_PLOT | NOCLICK_LAYER | NOCLICK_TICKLABEL |
NOCLICK_LAYERICON; GraphPage m_gp = Project.GraphPages("Graph1"); //OPJ にあるこのグラフは手動でカスタマイズ可能です。 //OPJ の Graph1 を GraphControl に加えます。 BOOL bb = m_gcCntrl.AttachPage(m_gp,
dwNoClicks); return true;     }      BOOL OnDestroy()     { //このディベロッパーで、OPJ を修正できるようにしています。          //この部分をエンドユーザ用に削除し、LT_execute(";doc -ss;exit;")を実行のみします。 bool bExitOrigin = MessageBox(GetSafeHwnd(), _L("Are you sure you want to exit
Origin?"),_L("Vertical Cursor Example"), MB_YESNO) == IDYES; if (bExitOrigin)     LT_execute(";doc -
ss;exit;"); //Origin を終了します。 else     LT_execute(";doc -m 1"); //Origin のメインウィンドウを表示します。 return true;     }      //virtual     //ダイアログの準備ができたら、ダイアログのサイズと位置を初期化する必要があります。      BOOL GetDlgInitSize(int& width, int& height)     { width = 1024;
height = 450; return true;     }     // ダイアログのサイズを変更する場合、ダイアログのそれぞれのコントロールのサイズと位置を再度初期化する必要があります。      BOOL OnDlgResize(int nType, int cx, int
cy)     { if ( !IsInitReady() )     return false; // MoveControlsHelper      _temp(this); // ダイアログのサイズを変更してそれがフリックしている場合、この行のコメントを外すことができます。
HTMLDlg::OnDlgResize(nType, cx, cy); //ダイアログで html コントロールを配置します。    if
( !IsHTMLDocumentCompleted() ) //HTML コントロールの状態を確認します。     return false;   RECT
rectGraph;          //GetGraphRECT はプライベート関数で JavaScript を呼び出して、グラフコントロールに関連付く四角形のボックスを作成します。//次のセクションの OriginC で、JavaScript を呼び出す方法を得ます。
if ( !GetGraphRECT(rectGraph))     return false; //HTML コントロールの上に GraphControl を重ねます。
m_gcCntrl.SetWindowPos(HWND_TOP, rectGraph.left,
rectGraph.top,RECT_WIDTH(rectGraph),RECT_HEIGHT(rectGraph), 0);    return true;     }
```

> **Note:** 手法 OnInitDialog() がトリガーにされる場合、グラフは、新しく作成したグラフコントロールに含まれます。

7. クラス HTMLandGraphDlg 内のダイアログメッセージをマップして、どのイベントを扱うのか、どの関数を呼び出すのかを指定します。

```
EVENTS_BEGIN_DERIV(HTMLDlg)      ON_INIT(OnInitDialog)      ON_DESTROY(OnDestroy)      ON_SIZE(OnDlgResize
)      ON_RESTORESIZE(OnRestoreSize) EVENTS_END_DERIV
```

Origin C で JavaScript を呼び出す

このセクションでは、OriginC で JavaScript を呼び出して、以下を行います。

- ダイアログのサイズを変更すると、グラフコントロールは中央パネルに沿って右に移動します。そして、同時に JavaScript を呼び出し、四角形のボックスを返します。
- 垂直線を移動すると、テキストボックスの X 値が更新されます。そして、JavaScript を呼び出して数字を更新します。

これらのセクションで、メンバー関数 GetScript()を使って、スクリプトエンジンインターフェースを得ます。この関数は、DHtmlControl クラスにあります。

1. 関数 GetGraphRect()をクラス HTMLandGraphDlg に追加して、JavaScript 関数 getGraphRECT()を呼び出します。これは 5 番目のセクションに書かれています。

```
private:     BOOL GetGraphRECT(RECT& gcCntrlRect) //JavaScript と呼び出して GraphControl の位置を得る関
数です。      {          if (!m_dhtml)     return false; Object jsscript =
m_dhtml.GetScript();    if(!jsscript) //返された COM オブジェクトが常に推奨されていることを確認しま
す。      return false;    string str = jsscript.getGraphRECT();     JSON.FromString(gcCntrlRect, str);
//文字列を構造体に変換します。 return true;      }
```

2. 関数 OnMoveVlineToUpdateHtml()を Origin C クラスに加えて、JavaScript で関数 lineMove()を呼び出します。

```
public:      void OnMoveVlineToUpdateHtml(double dVal)      {          Object jsscript =
m_dhtml.GetScript();          if(!jsscript)              return;          string strXValue = ftoa(dVal,
"*5*");//小数点桁数を 5 までにして、末尾のゼロを削除しま
す。          jsscript.lineMove(strXValue);      }
```

Origin C でプロトタイプを呼び出しできる JavaScript

このセクションでは、手法 OnEnterXvalToUpdateGraph()をダイアログクラス HTMLandGraphDlg に追加します。X 値を入力して Enter キーを押す際に垂直線を移動するため、このクラスは JavaScript 関数 enterXval()で呼び出します。

1. 手法と DECLARE_DISPATCH_MAP をダイアログクラスに追加します。

```
public:     DECLARE_DISPATCH_MAP       void OnEnterXvalToUpdateGraph(string
strXValue)      {         GraphPage gp = m_gcCntrl.GetPage();          if(!gp)     return; GraphLayer gl
= gp.Layers(0); if(!gl)     return; double xVal = atof(strXValue); if(NANUM == xVal)//数値を入力しません。     return; GraphObject vline; vline = gl.GraphObjects("Line"); //グラフの垂直線にアクセスします。 if(!vline)     return; vline.X = xVal;     }
```

2. OriginC クラスの手法を HTML ダイアログにマップします。

```
BEGIN_DISPATCH_MAP(HTMLandGraphDlg, HTMLDlg)     DISP_FUNCTION(HTMLandGraphDlg,
OnEnterXvalToUpdateGraph, VTS_VOID, VTS_STR) END_DISPATCH_MAP
```

Note:DISP_FUNCTION の詳細についてはこちらをご覧ください。

Origin C でプロトタイプを呼び出しできる LabTalk

最後に追加する項目があります。
プロパティダイアログの垂直線プロパティの設定をする際に、関数 line_move()を LabTalk で呼び出すスクリプトを書いたことを思い出してください。
OriginC にこの関数を追加して垂直線を移動する際のトリガーにします。

```
static HTMLandGraphDlg* s_pDlg = NULL;//ダイアログクラスを LabTalk から呼び出す必要があります。  //これは、LabTalk から呼び出される OC 関数です。 void line_move(double dVal) //直線を移動すると X 値を得て、JavaScript に値を渡します。 {    if ( s_pDlg ) //このダイアログで直線を移動したときのみのトリガー s_pDlg->OnMoveVlineToUpdateHtml(dVal); }
```

Note:ダイアログクラスで LabTalk の手法を呼び出すには、HTMLandGraphDlg オブジェクトを宣言します。

ダイアログの起動

ダイアログを起動する準備ができたら、

1. メイン関数を追加します。

```
void HTMLandGraphDlgEx() {     HTMLandGraphDlg dlg;     s_pDlg =
&dlg;     dlg.DoModalEx(GetWindow());     s_pDlg = NULL; }
```

Note: ProjectEvents.OGS.に書いた関数と同じ名前を付けます。

3. 全てのコードを保存して、ビルドします。
4. OPJ をダブルクリックして、直接ダイアログを立ち上げます。

18.2 ウェイトカーソル

waitCursor クラスは、マウスポインタを砂時計または処理を表すポインタに変更します。それは、Origin が処理に時間がかかるコードを実行していることを示すための視覚的な合図で、他の入力を受け無いようにします。*waitCursor* オブジェクトのインスタンスが作成されると、マウスポインタは処理を表すポインタに変わり、インスタンスが破棄されると、矢印のポインタに戻ります。

次のサンプルは、時間がかかる処理を行う関数です。最初に、*waitCursor* インスタンスを宣言し、作成します。作成されている間、マウスポインタは、処理を表すポインタに変わります。関数を終了すると、*waitCursor* のインスタンスは自動的に破棄され、マウスポインタが矢印のポインタに戻ります。

```
void myTimeConsumingFunction()

{

    waitCursor wc; // 宣言してウェイトカーソルを表示

    for( int i = 0; i < 10000; i++ )

    {

        if( 0 == (i % 100) )

            printf("i == %d¥n", i);

    }

}
```

次のサンプルは、上記の例に似ていますが、時間のかかる処理が完了する前に、関数を終了する機能を追加しました。早めに終了する機能は、ウエイトカーソルの *CheckEsc* メソッドを呼び出すことで実行されます。このメソッドは、ユーザが ESC キーを押すと True を返し、それ以外の場合は False を返します。

```
void myEscapableTimeConsumingFunction()

{

    waitCursor wc; // 宣言してウェイトカーソルを表示

    for( int i = 0; i < 10000; i++ )

    {

        if( 0 == (i % 100) )

            printf("i == %d¥n", i);

        if( wc.CheckEsc() )

            break; // ループを早く抜ける

    }

}
```

18.3 グラフからデータポイントを取得

Origin C の GetGraphPoints クラスは、グラフウィンドウの曲線からデータポイントを取得するのに使用されます。これは、仮想的なメソッドを持ち、メソッドをオーバーロードするためにそれから派生されます。

以下のサンプルは、グラフから 2 つのデータポイントを取得するし、*GetGraphPoints* クラスを使用する方法を示します。

```
GetGraphPoints mypts;



// true にセットすると、カーソルがデータプロットに沿って移動し、

// 曲線からデータポイントを取得

// false にセットすると、カーソルはデータプロットに沿って移動しません

// 画面上からデータポイントを取得

mypts.SetFollowData(true, dp.GetIndex());



// GraphLayer オブジェクト(gl)で指定した Graph からデータポイントを取得

int nPts = 2; // 取得するポイント数

mypts.GetPoints(nPts, gl);



// 取得したデータから x/y データとそのインデックスを取得

vector vx, vy;

vector<int> vnPtsIndices, vnPlotIndices;

if( mypts.GetData(vx, vy, vnPtsIndices, vnPlotIndices) == nPts )

{

        for(int ii = 0; ii < vx.GetSize(); ii++)

        {

                printf("point %d: index = %d, x = %g, y = %g, on plot %d¥n",

                        ii+1, vnPtsIndices[ii], vx[ii], vy[ii], vnPlotIndices[ii]+1);

        }

}
```

18.4 グラフにコントロールを追加する

ワークブックのオーガナイザや極座標グラフの上部にあるようにダイアログをページに接続する場合、**PageBase** クラスの **SetSplitters** メソッドを使います。

ダイアログバーをページに追加するには、**lpcszString** は、ダイアログクラスの名前とページウィンドウの位置(上下左右)を含む必要があります。**lpcszString** を NULL にセットすると、既存のダイアログバーを削除します。

次のサンプルは、グラフウィンドウにユーザ定義のダイアログを追加および削除する方法を示しています。

ユーザ定義のダイアログのクラス

```
#include <..¥Originlab¥DialogEx.h>

// OC_REGISTERED キーワードは、このクラスを見つけるために PageBase::SetSplitters メソッド

// を許可する必要がある

class OC_REGISTERED MyGraphPolarBar :public Dialog

{

public:

        // IDD_POLAR_CONTROL はダイアログのリソース ID

        // Odlg8 はダイアログリソース DLL ファイル名で、パスが見つからない場合

        // デフォルトのパスは  OriginC¥Originlab

        MyGraphPolarBar()

        :Dialog(IDD_POLAR_CONTROL, "Odlg8")

        {

        }



        BOOL CreateWindow(int nID, HWND hWnd)

        {

                int nRet = Dialog::Create(hWnd, DLG_AS_CHILD);

                HWND hWndThis = GetSafeHwnd();

                SetWindowLong(hWndThis, GWL_ID, nID);

                return nRet;

        }

};
```

グラフウィンドウのダイアログを追加または削除

```
void Page_SplittersControl(BOOL bShow = TRUE, int nPos = 2)
{
	Page pg = Project.Pages("Graph1");

	if( bShow )
	{
		int nPercent = 30;
		string strDlgClass = "MyGraphPolarBar"; // 上記ダイアログクラス

		string  strConfig;
		switch(nPos)
		{
		case 0:// 下
			strConfig.Format("r{%s}r[%s]", (string)nPercent+"%", strDlgClass);
			break;
		case 1:// 右
			strConfig.Format("c{%s}c[%s]", (string)nPercent+"%", strDlgClass);
			break;
		case 2:// 上
			strConfig.Format("r[%s]{%d}r", strDlgClass, nPercent);
			break;
		case 3:// 左
			strConfig.Format("c[%s]{%d}c", strDlgClass, nPercent);
			break;
		}
		pg.SetSplitters(strConfig);
	}
```

```
        else

                pg.SetSplitters(NULL);      // ページからダイアログバーを削除



}
```

19 外部リソースへのアクセス

Origin C は外部 DLL にアクセスすることができ、さらにオートメーション(COM)サーバの機能を使って、Origin 以外のアプリケーションにアクセスすることもできます。

19.1 サードパーティ製 DLL 関数にアクセスする

宣言

Origin C 関数は、C、C++、C++(.Net)、C#、Fortran コンパイラで作成した外部 DLL の関数を呼び出すことができます。これを行うには、ヘッダファイルの関数のプロトタイプを提供し、DLL ファイルに関数本体が含まれていることを Origin C に通知する必要があります。関数は、**myFunc.h** というヘッダファイルで宣言されているものとします。次のようにこれらの関数を呼び出したい場所で Origin C ファイルにこのファイルを**インクルード** します。

```
#include <myFunc.h> // ¥OriginC¥System folder 内

#include "myFunc.h" // Origin C コードと同じフォルダ内

#include "C:¥myFile.h" // 特定のパス内
```

DLL をロード

それから、Origin C に関数本体をリンクする場所を通知し、ヘッダーファイル **myFunc.h** 内の外部 DLL を呼び出す直前に次の Origin C **pragma** ディレクティブを含めます。DLL ファイルは **UserFunc.dll** とします。

```
#pragma dll(UserFunc) // Origin exe フォルダ内

#pragma dll(C:¥UserFunc) //特定パス内

#pragma dll(UserFunc, header) //h ファイルと同じフォルダ内

#pragma dll(UserFunc, system) //Windows システムフォルダ
```

Origin C コンパイラは次の 3 つの呼び出し方法をサポートしています。**__cdecl**(デフォルト), **__stdcall**, **__fastcall** です。これらの呼び出し方法は、引数がスタックに渡される順番を決定するだけでなく、呼び出している関数または呼び出された外部関数がスタックから引数を除去します。

Note: ファイル名に dll という拡張子を含める必要はありません。**pragma** ディレクティブの後のすべての関数宣言は、外部または特定の DLL からであると考えられます。この前提は、2 番目の**#pragma dll(filename)** ディレクティブが現れるか、ファイルの最後まで続きます。

バージョン制御

外部 dll を正常に動作させるには、32bit の dll は Origin の 32bit 版で行う必要があります（64bit 版も同様）。**#ifdef _OWIN64** は、現在の Origin が 32/64bit どちらのバージョンであるか確認するため、どちらのバージョンの dll がロードされたか決定するために使用されます。例えば、

```
#ifdef      _OWIN64

#pragma dll(UserFunc_64, header)

#else
```

```
#pragma dll(UserFunc, header)

#endif     //_OWIN64
```

サンプル

外部 DLL にアクセスする方法のサンプルは、Accessing SQLite Database です。Origin C で C DLL、Matlab、Fortran DLL からの関数の呼び出し方法を示すサンプルプロジェクトがあります。これらは、Origin の*\Samples\Origin C Examples\Programming Guide* フォルダの *Calling Fortran*, *Calling MATLAB DLL*, *Calling C DLL* サブフォルダにあります。

19.1.1 GNU Scientific Library を呼び出す

ここでは、OriginC で GSL を使用する方法を紹介します。まず、GSL dll が必要です。GSL dll のビルド方法はここを確認するか、DLL を http://gnuwin32.sourceforge.net/packages/gsl.htm からダウンロードします。2 つの DLL (libgsl.dll と libgslcblas.dll) が必要で、OriginC ファイルと同じフォルダに保存してください。例えば、c:\oc\ フォルダに保存します。ダウンロードした dll を使用する場合、バージョンの問題に注意してください。

libgsl.dll

メインの dll です。

libgslcblas.dll

この dll は、libgsl.dll に必要です。

Origin C で libgsl.dll を使用するために、gsl 関数のプロトタイプを提供するヘッダファイルが必要です。必要に応じて、GSL ヘッダファイルから必要なプロトタイプ/定義をコピーして移動できます。たとえば、**ocgsl.h** を呼び出し、c:\oc\ に作成します。

ocgsl.h

```
// dll をロードするとき、正しいバージョンをロードする必要があるので、

// 上の "バージョンの問題" のリンクを確認

#pragma dll(libgsl, header)

// これは OC の特別なプラグマで、

// libgsl.dll はこのファイルと同じ場所であることを示すキーワード



#define GSL_EXPORT // OC では、これは不要なので空にする



// gsl 関数プロトタイプをここで直接検索してコピー可能



GSL_EXPORT double gsl_sf_zeta_int (const int n);

 GSL_EXPORT int gsl_fit_linear (const double * x, const size_t xstride,
```

```
                                const double * y, const size_t ystride,

                                const size_t n,

                                double * c0, double * c1,

                                double * cov00, double * cov01, double * cov11,

                                double * sumsq);
```

次のサンプル OC ファイルでは、gsl_sf_zeta_int と gsl_fit_linear の呼び出し方を示しています。

test_gsl.c

```
s#include <Origin.h>

#include "ocgsl.h"



// GSL の Riemann Zeta 関数を使用したサンプル

void gsl_test_zeta_function()

{

        double result1 = gsl_sf_zeta_int(2);

        double result2 = pi*pi/6;



        printf("Zeta(2) = %f¥n", result1);

        printf("pi^2/6  = %f¥n", result2);

}



// GSL の線形フィットを使用したサンプル

void gsl_test_linear_fit(int npts = 10)

{

        vector vx(npts), vy(npts);

        const double ds = 2, di = 10;
```

```
        for(int ii=0; ii<npts; ++ii)
        {
                vx[ii] = ii;
                vy[ii] = ii*ds + di + (rand()%100-50)*0.05;
        }

        for(ii=0; ii<npts; ++ii)
                printf("%.2f¥t%.2f¥n", vx[ii], vy[ii]);

        double c0, c1, cov00, cov01, cov11, sumsq;

        gsl_fit_linear(vx, 1, vy, 1, npts, &c0, &c1, &cov00, &cov01, &cov11, &sumsq);

        printf("Slope=%f, Intercept=%f", c1, c0);
}
```

フィット関数内に GSL を使用

このサンプルでも、フィット関数内に gsl 関数を使用する方法を示しています。

GSL 関数を使用する上での注意

Origin C は、構造体変数を返す外部関数をサポートしていないので、このようなデータを返す関数は Origin C では使用できません。例えば、

```
gsl_complex gsl_complex_add (gsl_complex a, gsl_complex b)
```

データの gsl_complex タイプを返し、gsl_complex は次のように定義されます。

```
typedef struct
{
    double dat[2];
}gsl_complex;
```

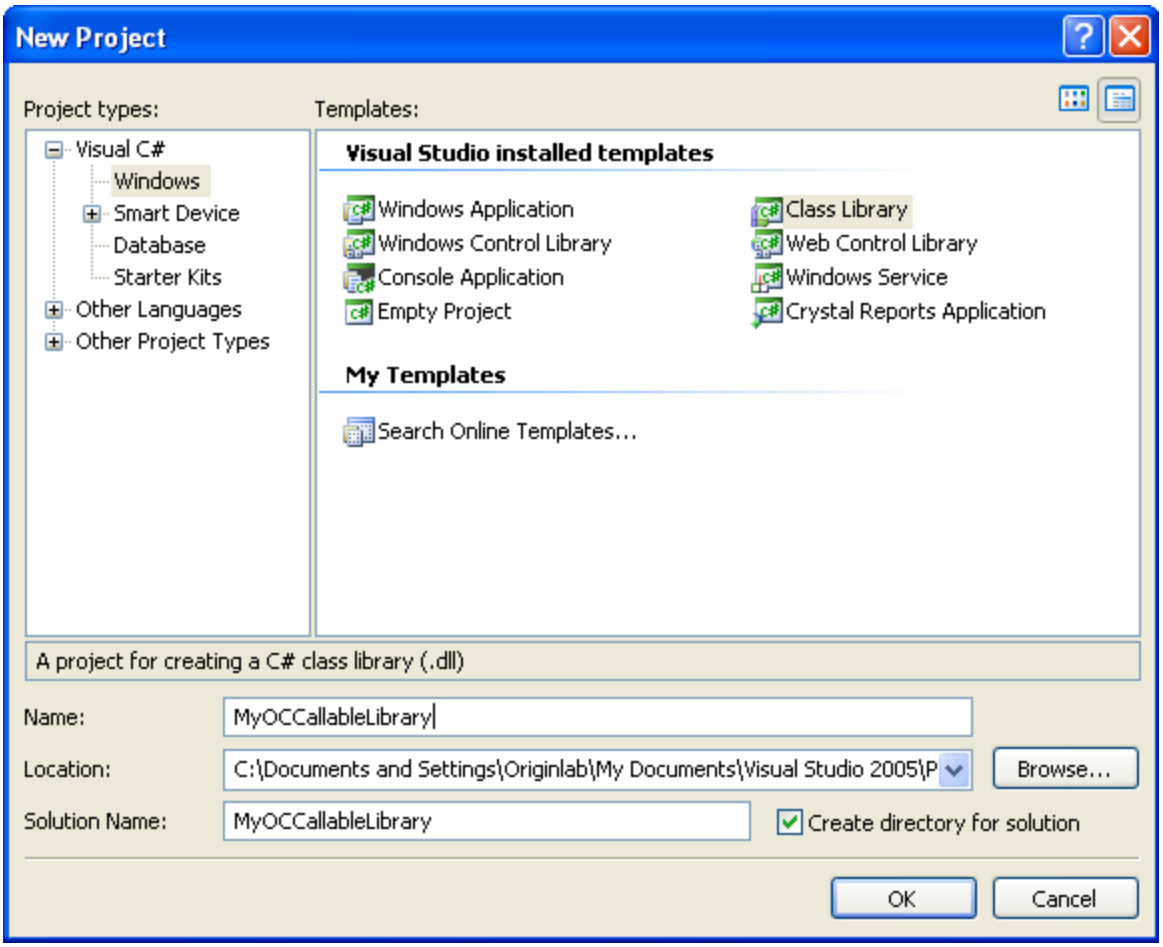
19.1.2 C++(.Net) と C# DLL にアクセスする

この章では、Origin C で C++(.Net) や C# から作成された DLL にアクセスする方法を紹介しています。

C# クラス DLL にアクセスする

Microsoft Visual Studio 2005 で C#の DLL を作成し、Origin C でそれにアクセスする方法を紹介しています。この DLL は、プロパティと関数をクラスに提供するものです。関数 Sum は、Origin C ベクトルから C#関数にデータ配列を渡す方法を示しています。

1. Microsoft Visual Studio 2005 で、メニューからファイル -> 新規 -> プロジェクト...を選択し、新しいプロジェクトダイアログで、次のように設定を選択してください。



2. 次のコードを cs ファイルにコピーします。

```
using System;

using System.Collections.Generic;

using System.Text;

using System.Runtime.InteropServices;
```

```
namespace temp
{
    [Guid("4A5BFDFA-7D41-49d1-BB57-C6816E9EDC87")]
    public interface INet_Temp
    {
        double Celsius{ get; set; }
        double Fahrenheit{ get; set; }

        double GetCelsius();
        double GetFahrenheit();

        double Sum(object obj);
    }
}

namespace temp
{
    [Guid("2AE913C6-795F-49cc-B8DF-FAF7FBA49538")]
    public class NET_Temperature :INet_Temp
    {
        private double celsius;
        private double fahrenheit;

        public NET_Temperature()
        {
        }
```

```
    public double Celsius
    {
        get{ return celsius; }
        set { celsius = value; fahrenheit = celsius + 273.5;  }
    }
    public double Fahrenheit
    {
        get { return fahrenheit; }
        set { fahrenheit = value; celsius = fahrenheit - 273.5; }
    }

    public double GetCelsius()
    {
        return celsius;
    }
    public double GetFahrenheit()
    {
        return fahrenheit;
    }

    public double Sum(object obj)
    {
        double[] arr = (double[])obj;
        double sum = 0.0;
        for (int nn = 0; nn < arr.Length; nn++)
        {
```

```
                sum += arr[nn];

            }

            return sum;

        }

    }

}
```

3. メニューから、ツール -> GUID の作成 を選択して、開いたダイアログで Registry Format を選択して New GUID をクリックし、Copy ボタンをクリックします。この GUID を貼り付けて、上述のコードの[Guid("...")] 内のものを削除します。コード内の 2 番目の GUID を置き換えるためには、このアクションを再度行います。
4. プロジェクト上で右クリックして、プロパティを選択し、プロパティページを開きます。アプリケーションタブを開き、アセンブリ情報ボタンをクリックします。そして、アセンブリを COM 参照可能にするのチェックボックスにチェックを付けます。32 bit 版の場合、ビルドタブを開き、COM 相互運用機能の登録のチェックボックスをチェックします。64 bit 版の場合、ビルドイベントタブを開き、次のコマンドラインをコピーしてビルド前に実行するコマンドラインテキストボックスに貼り付けます。F6 キーを押してソリューションをビルドします。(エラーが出る場合は、Visual Studio を「管理者として実行」で起動しなおしてください)

```
"%Windir%¥Microsoft.NET¥Framework64¥v4.0.30319¥regasm" "$(TargetPath)" /CodeBase
```

5. Origin を開き、コードビルダを開いて、新しい C ファイルに次の Origin C コードをコピーします。

```
void access_DLL()

{

        Object obj = CreateObject("temp.NET_Temperature");



        obj.Celsius = 0; // プロパティにアクセス

        out_double("", obj.GetFahrenheit()); // 関数にアクセス



        obj.Fahrenheit = 300;

        out_double("", obj.GetCelsius());
```

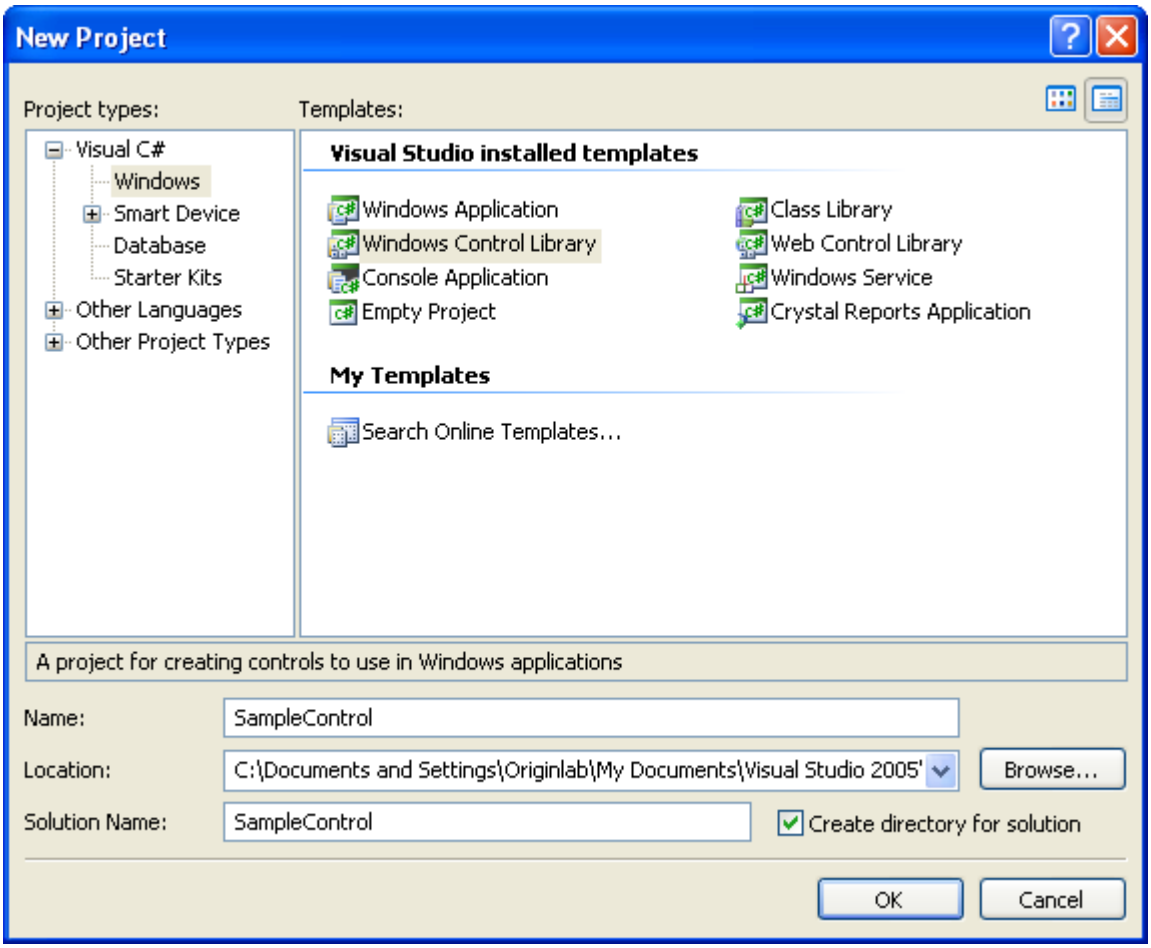
```
        vector vec;

        vec.Data(1,10,1);

        _VARIANT var = vec.GetAs1DArray();

        out_double("", obj.Sum(var));

}
```

C# と C++ リソース DLL にアクセスする

C#による ActiveX コントロールを作成し、Origin C 内のダイアログで使用する方法を示しています。

ステップ 1～7 では、C# ActiveX コントロールを Microsoft Visual Studio 2005 で作成する方法を示しています。

1. Microsoft Visual Studio 2005 を起動し、メニューからファイル -> 新規 -> プロジェクト... を選択して、新しいプロジェクトダイアログを開きます。そして、次のように設定して OK ボタンをクリックします。



2. UserControl.cs ファイルに次のコードをコピーして protected override function を追加し、色塗りの制御と public function を追加して境界の制御をします。

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Text;
using System.Windows.Forms;
using System.Runtime.InteropServices;


namespace SampleControl
{
    [Guid("A31FE123-FD5C-41a1-9102-D25EBD5FDFAF"),
    ComSourceInterfaces(typeof(UserEvents)),
    ClassInterface(ClassInterfaceType.None),]
    public partial class UserControl1 :UserControl, UserControl1Interface
    {
        public UserControl1()
        {
            InitializeComponent();
        }


        protected override void OnPaint(PaintEventArgs pe)
        {
            Brush brush = new SolidBrush(Color.Beige);


            pe.Graphics.FillRectangle(brush, ClientRectangle);
        }
```

```
        public void SetBorder(bool bSet)
        {
            this.BorderStyle = bSet ?BorderStyle.FixedSingle :
                        BorderStyle.None;
            Refresh();
        }
    }

    //設定制御のためのインターフェースを宣言
    [Guid("CCBD6133-813D-4dbb-BB91-16E3EFAE66B0")]
    public interface UserControl1Interface
    {
        void SetBorder(bool bSet);
    }
}
```

3. メニューからツール -> GUID の作成を選択して、GUID の作成ダイアログを開き、Registry Format ラジオボタンを選択します。New GUID と Copy ボタンをクリックして新しく作成された GUID をコピーします。新しい GUID を使用して上述のコード [Guid("...")] 内のものと置き換えます。
4. UserControl1.cs[Design] タブを選択し、プロパティウィンドウでイベントボタン をクリックし、スクロールバーをドラッグして MouseClick をダブルクリックしてマウスクリックイベント関数を UserControl.cs ファイルに追加します。同じ方法で、マウスダブルクリックイベントを追加します。
5. UserControl1I.cs ファイルでは、UserControl1 クラス外部に次のコードをコピーします。

```
//イベントのためのデリゲートを宣言

public delegate void MouseAction(int x, int y);
```

6. UserControl1 クラスで、次のインプリメントを追加します。

```
public event MouseAction OnUserClick;

public event MouseAction OnUserDbClick;



private void UserControl1_MouseClick(object sender, MouseEventArgs e)

{

      if (OnUserClick != null)

      {

          OnUserClick(e.X, e.Y);

      }

}



private void UserControl1_MouseDoubleClick(object sender, MouseEventArgs e)

{

      if (OnUserDbClick != null)

      {

          OnUserDbClick(e.X, e.Y);

      }

}
```

7. UserControl1 クラス外部で、次のインターフェースを追加します。

```
//イベントのためのインターフェースを宣言

[Guid("DA090A6F-FFAC-4a39-ACD3-351FA509CA86"),

    InterfaceType(ComInterfaceType.InterfaceIsIDispatch)]

public interface UserEvents

{

    [DispIdAttribute(0x60020001)]

    void OnUserClick(int x, int y);
```

```
    [DispIdAttribute(0x60020002)]
    void OnUserDbClick(int x, int y);
};
```

8. Visual Studio 2008 でリソースのみの DLL を作成する方法は次のドキュメントを確認してください。ダイアログビルダ:単純な Hello World ダイアログ:Visual Studio 2008 でリソースのみの DLL を作成する
9. Origin C によって、ダイアログで ActiveX コントロールを使用し、表示します。Origin のコードビルダで、c ファイルを作成し、次のコードをコピーします。

```
//ダイアログと Ids のコントロール。実際の利用に応じてダイアログ id は変更する必要がある。
#define IDD_DIALOG1                     101
#define IDC_DOTNET                                      1000

//コントロールから公開されたイベント
#define    ON_INTEROP_CLICK(_idCntl, _ocFunc)
ON_ACTIVEX_EVENT(0x60020001, _idCntl, _ocFunc, VTS_CTRL VTS_I4 VTS_I4)

#define    ON_INTEROP_DBLCLICK(_idCntl, _ocFunc)
ON_ACTIVEX_EVENT(0x60020002, _idCntl, _ocFunc, VTS_CTRL VTS_I4 VTS_I4)


class CDotNetComInteropDlg :public Dialog
{
public:
        CDotNetComInteropDlg();

        EVENTS_BEGIN
```

```
        ON_INIT(OnInitDialog)
        //公開されたイベントのためのイベントハンドラ
        ON_INTEROP_CLICK(IDC_DOTNET, OnClick)
        ON_INTEROP_DBLCLICK(IDC_DOTNET, OnDblClick)
        EVENTS_END

        BOOL OnClick(Control ctrl, int x, int y)
        {
                printf("Clicked at (%d,%d)¥n", x,y);
                return true;
        }

        BOOL OnDblClick(Control ctrl, int x, int y)
        {
                printf("DblClicked at (%d,%d)¥n", x,y);
                return true;
        }

        BOOL OnInitDialog();

        Control m_ctrlDotNet;
};

// ここでは DLL ファイルのパスを指定しない。DLL ファイルは
// 現在の c ファイルと同じパスにあると仮定
```

```
CDotNetComInteropDlg::CDotNetComInteropDlg()
        :Dialog(IDD_DIALOG1, "DialogBuilder.dll")
{
        InitMsgMap();
}



BOOL CDotNetComInteropDlg::OnInitDialog()
{
        //[Guid("A31FE123-FD5C-41a1-9102-D25EBD5FDFAF")]
        GUID guid = {0xA31FE123, 0xFD5C, 0x41a1,
                {0x91, 0x02, 0xD2, 0x5E, 0xBD, 0x5F, 0xDF, 0xAF}};

        RECT rect = {20,20,200,100};
        if(m_ctrlDotNet.CreateActiveXControl(guid, WS_CHILD|WS_VISIBLE,
                rect, GetSafeHwnd(), IDC_DOTNET))
        {
                //DotNet コントロールのインターフェースを使用して
                //初期化をコントロール
                Object ctrlObj = m_ctrlDotNet.GetActiveXControl();
                ctrlObj.SetBorder(true);
        }
        return TRUE;
}

void Launch_CDotNetComInteropDlg()
{
```

```
        CDotNetComInteropDlg dlgDotNet;

        dlgDotNet.DoModal();

}
```

19.1.3 外部 DLL から Python にアクセスする

Origin C では、直接 Python を呼び出す方法はありません。Origin C 内で Python コードを再利用したい場合、まず DLL に Python 関数をラップした後、関数を DLL で Origin C コードに公開することをお勧めします。この章の以下の節では、手順を追ってこれを説明するサンプルを示します。

Note: Origin 2015 から python を Origin で実行でき(コマンドラインと.py ファイルの両方をサポート)、**PyOrigin** モジュールを使用して Python から Origin にアクセス可能です。詳細は、Python.chm を確認してください。

実行環境

このサンプルは以下の環境で行ったものです。

1. Windows 7
2. Python 3.3.5 (Python の設定は完了していることを仮定)
3. Visual Studio 2012

Python ファイル

以下で定義される Python 関数を Origin C で再利用します。

```
def SayHello():
    print("Welcome to use Python in your Origin C programs!")#実際はこの文字列は Origin C で表示されない
    return 12345


def Add(a, b):
    return a + b


def Sub(a, b):
    return a - b


def Mult(a, b):
    return a * b
```

```
def Div(a, b):
    return a / b

def Mod(a, b):
    return a % b
```

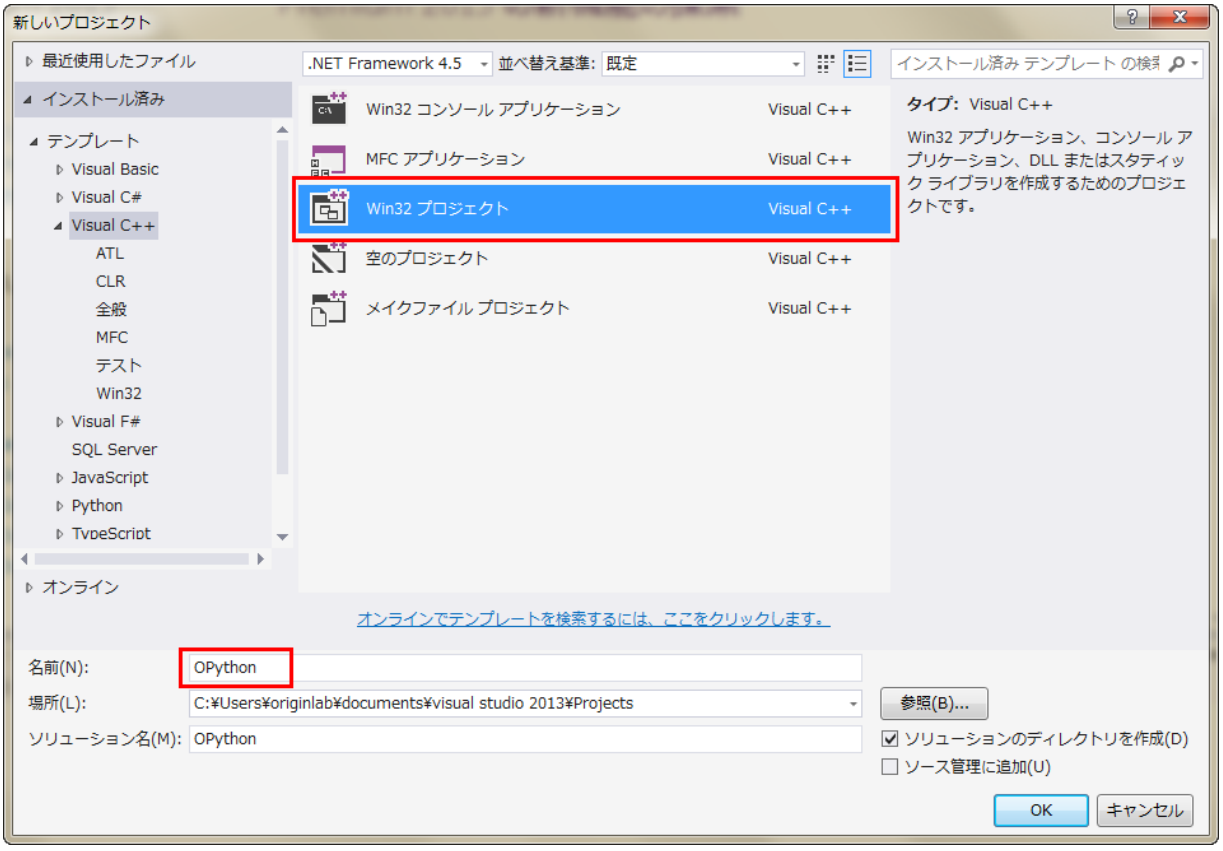
1. 最初に Python のインストールディレクトリを開き、このディレクトリ下に Python ファイル"myLib.py"を作成します。メモ帳等のテキストエディタを使用して新しく作成したファイルを開いて、このファイルに上述の Python コードを貼り付け、保存します。
2. コマンドプロンプト (cmd.exe) を実行し、現在のパスを Python がインストールされているディレクトリに切り替えます。
3. 以下のコマンドを実行して、作成した Python ファイルをコンパイルします。

```
python -m py_compile myLib.py
```

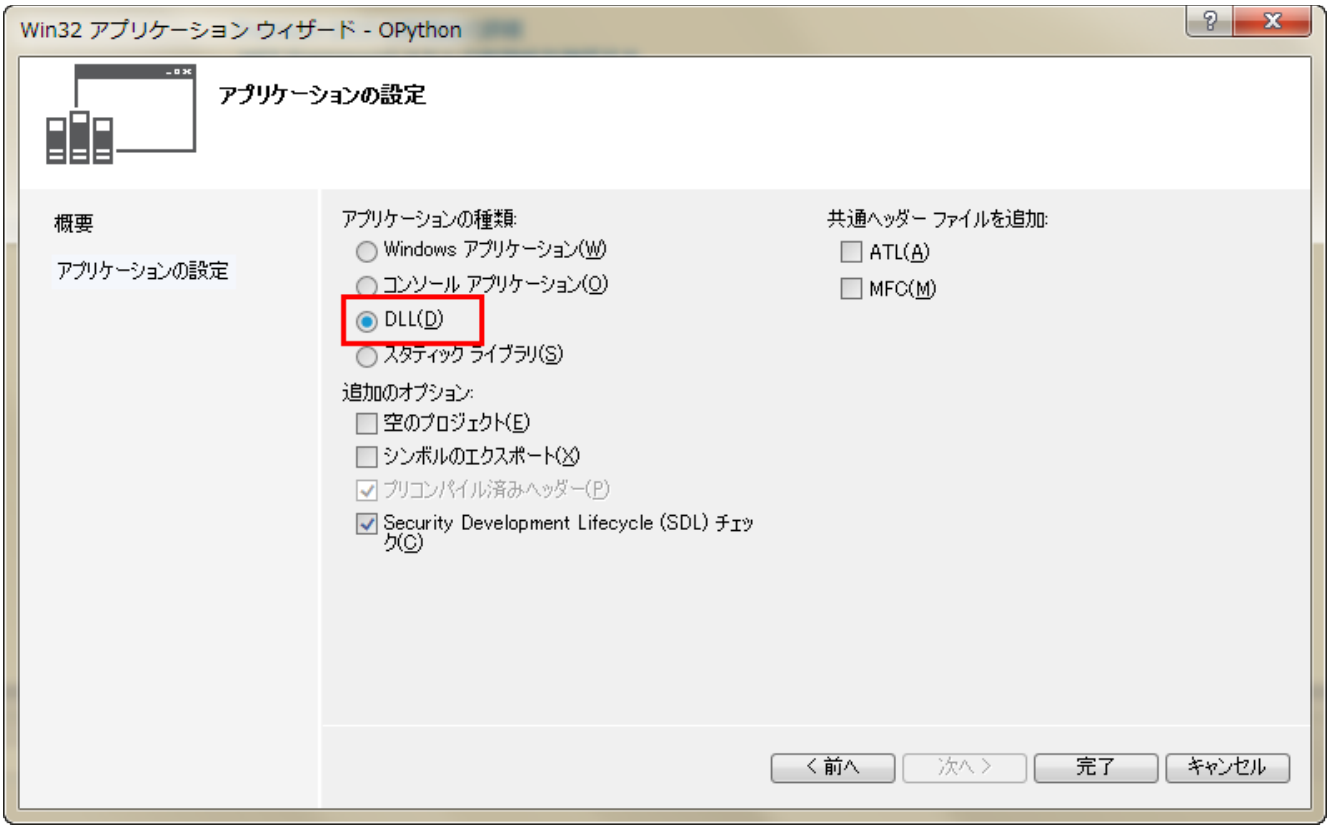
成功したら、デフォルトで pyc ファイルが<Python インストールディレクトリ>/__pycache__/ フォルダ下に作成されます。

組み込み DLL

1. Visual Studio 2012 を開始し、OPython という名前の新しい Win32 プロジェクトを作成します。



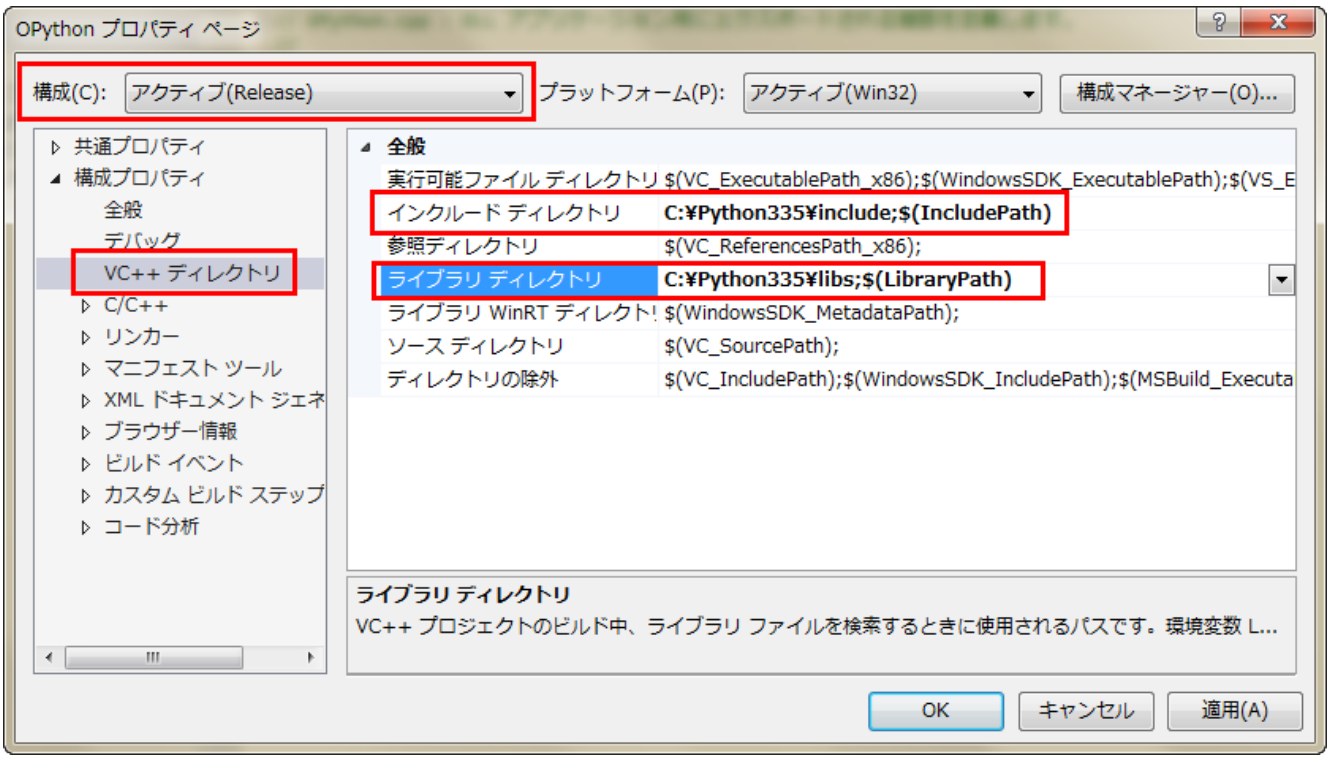
2. OK をクリックし、アプリケーションの種類を DLL にして完了をクリックしてプロジェクトを作成します。



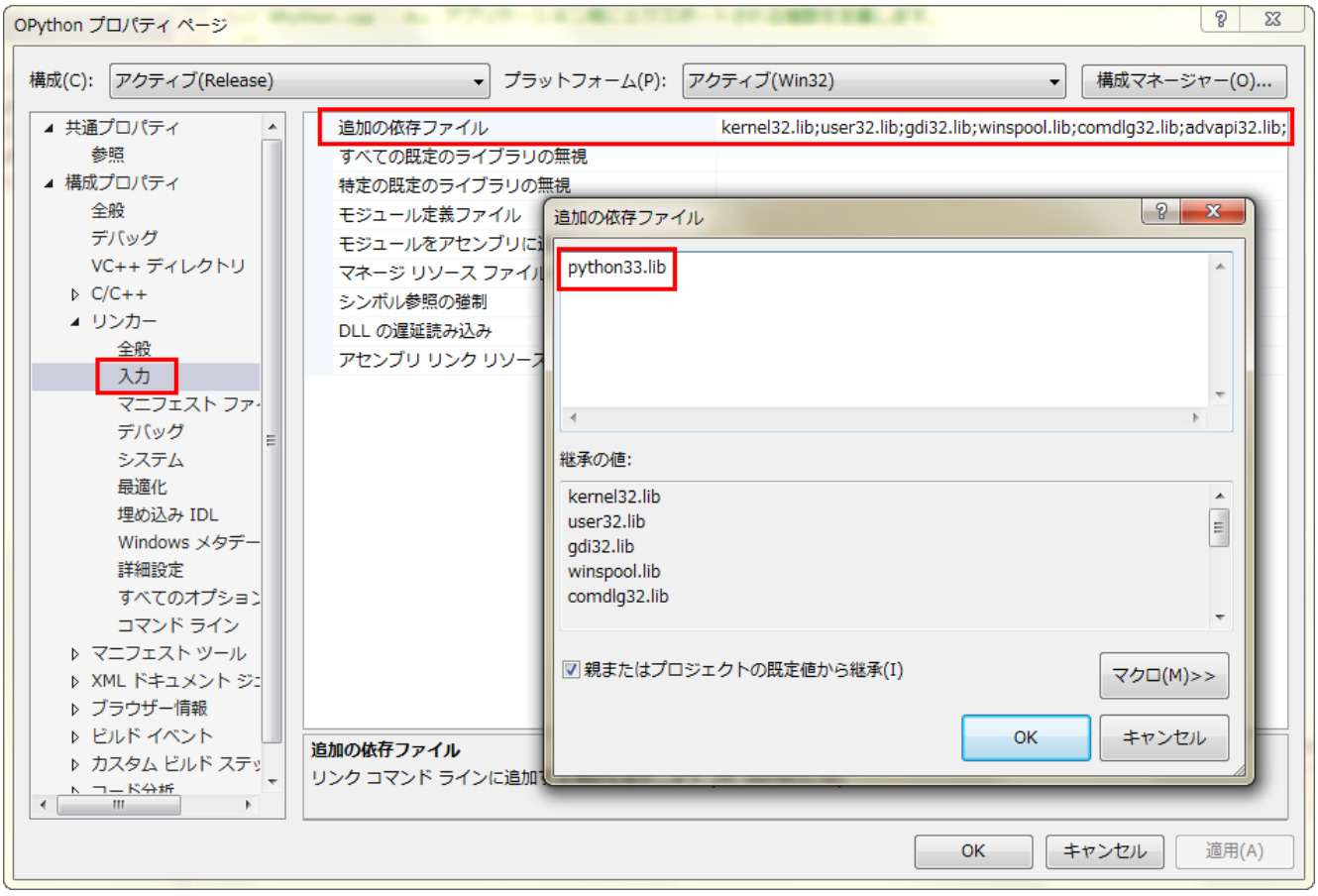
3. **ソリューション構成**を **Release** に設定します。プロジェクトを右クリックして、プロパティを選択し、プロパティダイアログを開きます。

   Note:32-bit DLL を作成したので、32-bit 版の Origin で使用できます。

4. プロパティダイアログで、左パネルから**構成プロパティ**:**VC++ ディレクトリ**を選択し、右パネルで**インクルードディレクトリ**と**ライブラリディレクトリ**それぞれに Python ヘッダファイルパスとライブラリパスを追加します。



5. 左パネルから**構成プロパティ:リンカー:入力**を選択し、右パネルで**追加の依存ファイル**に Python ライブラリ **python33.lib** を追加します。他の設定はそのままにして、OK をクリックします。



6. プロジェクトにヘッダファイル OPython.h を追加して以下のコードを貼り付けます。

```
#ifndef _OPYTHON_H_

#define  _OPYTHON_H_



#ifdef   _OPYTHON_CPP_

        #define    OP_API  __declspec(dllexport) //use in VC

#else

        #define    OP_API                                                    //OC で使用

        #pragma dll(OPython) //この行は重要です。OC で使用するとき、Originは、この DLL を検索すること
で関数本体を探します。Note: dll で生成される OPythonDLL は拡張名なしです。

#endif



OP_API  int          PY_SayHello();



OP_API  float PY_Add(float a, float b);



OP_API  float PY_Sub(float a, float b);



OP_API  float PY_Mult(float a, float b);



OP_API  float PY_Div(float a, float b);



OP_API  float PY_Mod(float a, float b);



#endif   //_OPYTHON_H_
```

7. プロジェクトを作成すると自動で生成される OPython.cpp を開きます(ない場合作成します)。元のコードを以下のコードで置き換えます。

```
#include "stdafx.h"

#define  _OPYTHON_CPP_ //VC(DLL を作成したとき) または OC(DLL を使用したとき)に OPython.h がインクルードされるかどうかを確認するためにこのマクロを使用

#include "OPython.h"

#include <Python.h>



class PythonManager

{

public:

        PythonManager(); //python 環境を初期化

        ~PythonManager(); //python 環境を削除

};



PythonManager::PythonManager()

{

        Py_Initialize();

}



PythonManager::~PythonManager()

{

        Py_Finalize();

}



OP_API  int          PY_SayHello()

{

        PythonManager pm;



        PyObject* pModule;
```

```
        PyObject* pFunc;
        //python 関数呼び出し、パラメータなし
        int nRet = 0;
        pModule = PyImport_ImportModule("myLib");
        if ( NULL == pModule )
                return nRet;

        pFunc = PyObject_GetAttrString(pModule, "SayHello");
        if ( pFunc )
        {
                PyObject* pRet = NULL;
                pRet = PyEval_CallObject(pFunc, NULL);

                PyArg_Parse(pRet, "i", &nRet);
        }
        return nRet;
}

static   float      _call_float_float(LPCSTR lpcszFunc, float a, float b)
{
        PythonManager pm;

        PyObject* pModule;
        PyObject* pFunc;
        //python function 関数呼び出し、複数パラメータ
        float fRet = 0;
        pModule = PyImport_ImportModule("myLib");
```

```
        if ( NULL == pModule )
                return fRet;

        pFunc = PyObject_GetAttrString(pModule, lpcszFunc);
        if ( pFunc )
        {
                PyObject* pParams = NULL;
                pParams = PyTuple_New(2);              //パラメータをおくタプルを作成

                PyTuple_SetItem(pParams, 0, Py_BuildValue("f", a));
                PyTuple_SetItem(pParams, 1, Py_BuildValue("f", b));
                PyObject* pRet = NULL;
                pRet = PyEval_CallObject(pFunc, pParams);

                PyArg_Parse(pRet, "f", &fRet);
        }
        return fRet;
}
OP_API  float PY_Add(float a, float b)
{
        return _call_float_float("Add", a, b);
}

OP_API  float PY_Sub(float a, float b)
{
        return _call_float_float("Sub", a, b);
}
```

```
OP_API  float PY_Mult(float a, float b)
{
        return _call_float_float("Mult", a, b);
}

OP_API  float PY_Div(float a, float b)
{
        return _call_float_float("Div", a, b);
}

OP_API  float PY_Mod(float a, float b)
{
        return _call_float_float("Mod", a, b);
}
```

8. モジュール定義ファイル OPython.def をプロジェクトに追加し、元のコードを以下のコードに置き換えます。

```
LIBRARY "OPython"

EXPORTS

                ;Functions to export
                PY_SayHello
                PY_Add
                PY_Sub
                PY_Mult
                PY_Div
                PY_Mod
```

9. OPython.dll という DLL を作成するためにソリューションをビルドします。

DLL を使用

以下のコードは、上で作成した DLL を Origin C で使用する方法を示します。

1. 作成した DLL(OPython.dll)をコピーして、Origin のインストールディレクトリに貼り付けます。
2. 作成したヘッダファイル(OPython.h)を次のフォルダにコピーします:**<ユーザファイルフォルダ>/OriginC/**
3. 32bit 版 Origin を起動し、コードビルダを開きます。新しい C ファイルを作成し、以下のコードで置き換えます。

```
#include <Origin.h>

#include "OPython.h" //このヘッダファイルをインクルードすること



int test_Python()

{

        int nRet = PY_SayHello();



        float c = PY_Add(2, 4);



        float d = PY_Div(2, 3);



        return 0;

}
```

4. コードビルダで上のコードをコンパイルし、以下の行を **LabTalk コンソール**(コードビルダのメニューから表示:LabTalk コンソールを選択)で実行します。

```
test_Python;
```

備考

このページのサンプルでは、Python で記述された関数を再利用するシンプルな DLL を作成します。大量データを Python で処理して Origin と Python 間でデータを交換したい場合、バッファが推奨されます。Origin の列からデータを渡すとき、データを持つベクターを、OriginC からパラメータをポインタ(double* pBuffer) で受ける DLL 関数に渡します。このバッファに基づく変更はすべて即座に Origin C 内のベクターに反映されます。

19.2 外部アプリケーションにアクセスする

Microsoft Component Object Model(COM)は、アプリケーションがバイナリソフトウェアコンポーネントから構築されるソフトウェアアーキテクチャで、ソフトウェアの開発がより簡単で効率よく行うことができます。

Origin は、COM クライアントプログラミングの機能を提供しており、**OriginPro** でのみサポートされています。このメカニズムは、すべての COM(オートメーションサーバ)オブジェクトを表す **オブジェクト**タイプを使用します。すべての COM オブジェクトは 2 つの方法で初期化することができます。

```
//CreateObject メソッドによる

Object oExcel;

oExcel = CreateObject("Excel.Application");



//既存の COM オブジェクトからの初期化による



Object oWorkBooks;

oWorkBooks = oExcel.Workbooks;
```

Origin C は、**Matlab** にアクセスするクラスも提供しており、これを使って Origin と Matlab 間の通信を可能にします。

```
#include <Origin.h>

#include <externApps.h> //MATLAB クラスに必要なヘッダ

void test_Matlab()

{

    Matlab matlabObj(true);

    if(!matlabObj)

    {

        out_str("No MATLAB found");

        return;

    }



    //ma という 3x5 の行列を定義

    string strRet;

    strRet = matlabObj.Execute("ma=[1 2 3 4 5;
```

```
        4 5 6 7 8;10.3 4.5 -4.7 -23.2 -6.7]");

    out_str(strRet);// MATLAB から str を表示



    // 行列を Origin 行列に配置

    MatrixLayer matLayer;

    matLayer.Create();

    Matrix mao(matLayer);



    //MATLAB の行列(ma)を Origin の mao 行列に変換

    BOOL bRet = matlabObj.GetMatrix("ma", &mao);

}
```

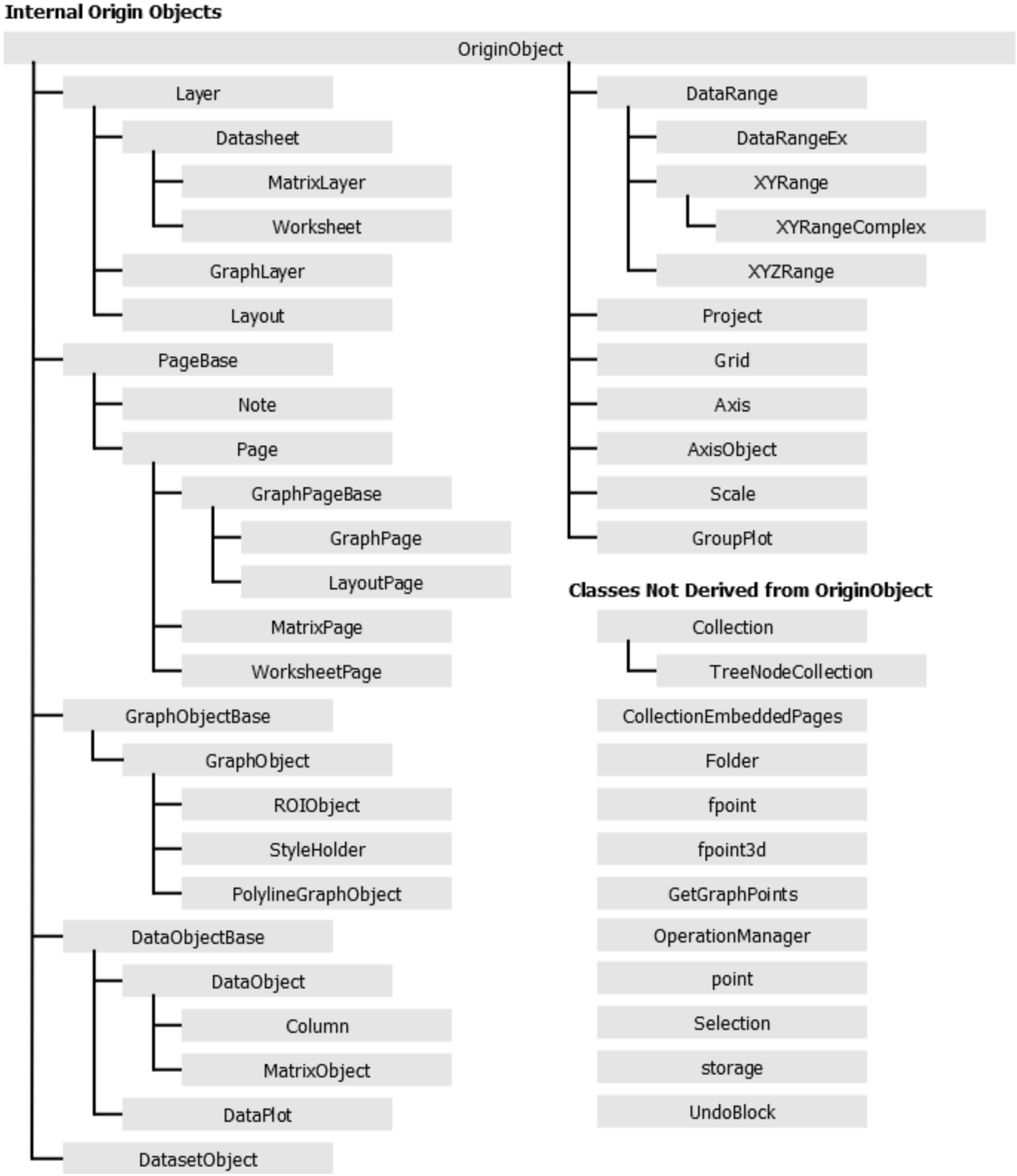
Origin C での COM クライアントプログラミングは、MS Office アプリケーションとデータのやりとりを行うのに使用できます。Excel ワークシートからデータを読み、Origin でグラフを作成し、それを Word 文書に配置するサンプルコードを用意しています。これは、Origin の *Samples\COM Server and Client\MS Office\Client* サブフォルダにあります。

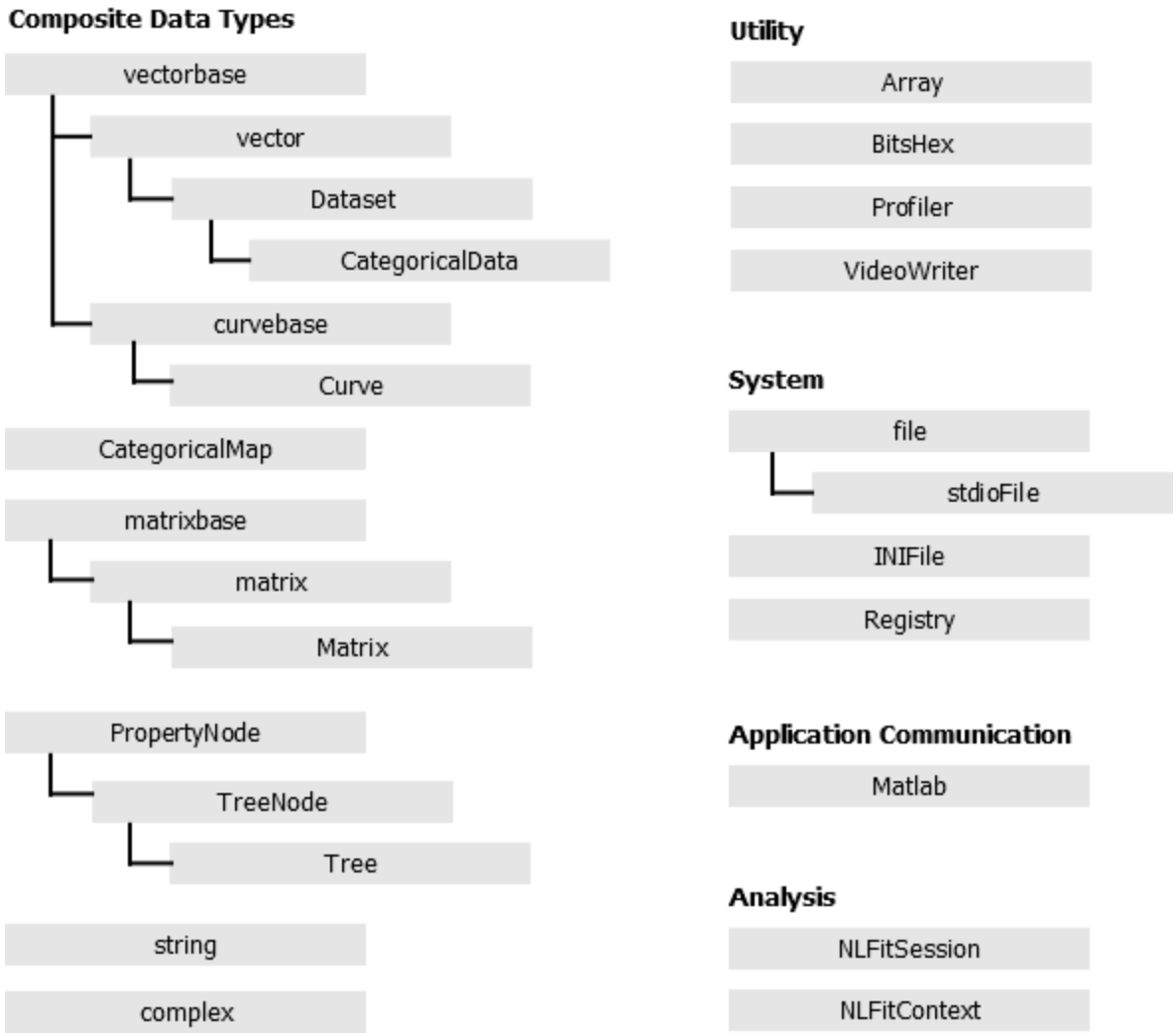
Origin C の COM プログラミングは、ADO(ActiveX Data Object)にアクセスしてデータベースと接続するのに使用することができ、SQL と Access データベースをワークシートにインポートし、データの修正を行った後データベースに戻すサンプルファイルがあります。詳細は、Origin の *Samples\COM Server and Client\ADO\Client* サブフォルダの ADOSample.c ファイルをご覧ください。

20 リファレンス

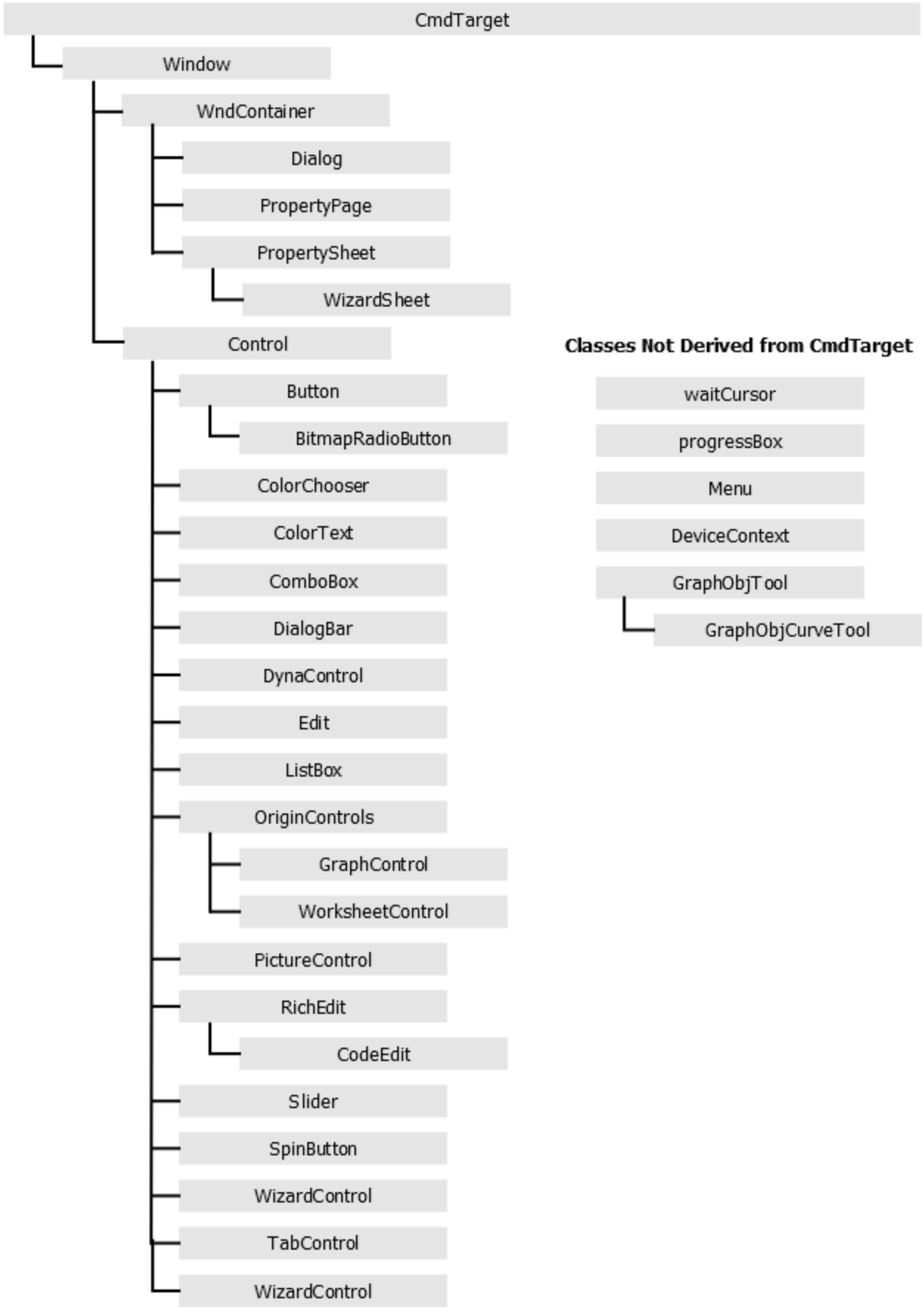
20.1 クラスの階層

次の図は、組込みの Origin C クラスの階層を示しています。




Composite Data Types
vectorbase
vector
Dataset
CategoricalData
curvebase
Curve
CategoricalMap
matrixbase
matrix
Matrix
PropertyNode
TreeNode
Tree
string
complex
Utility
Array
BitsHex
Profiler
VideoWriter
System
file
stdioFile
INIFile
Registry
Application Communication
Matlab
Analysis
NLFitSession
NLFitContext

User Interface Controls



Classes Not Derived from CmdTarget

- waitCursor
- progressBox
- Menu
- DeviceContext
- GraphObjTool
  - GraphObjCurveTool

20.2 コレクション

Collection クラスは、同じ型の複数のオブジェクトを保持するテンプレートを提供します。Origin C では、Collection のインスタンスは読み取り専用です。コレクションに項目を追加したり、削除することはできません。

コレクションのインスタンスであるデータメンバを含む Origin C クラスは数多くあります。例えば、Project クラスは WorksheetPages という名前のデータメンバを含みます。WorksheetPages データメンバは、プロジェクト内の既存の WorksheetPage オブジェクトすべてのコレクションです。

下表は、コレクションとコレクション内の項目タイプを含むクラスに沿って Origin C のコレクションのすべてを一覧表示しています。

クラス	データメンバー	コレクション
Folder	Pages	PageBase
Folder	SubFolders	Folder
GraphLayer	DataPlots	DataPlot
GraphLayer	StyleHolders	StyleHolder
GraphPage	Layers	GraphLayer
Layer	GraphObjects	GraphObject
MatrixLayer	MatrixObjects	MatrixObject
Page	Layers	Layer
Project	DatasetNames	string (loose and displayed datasets)
Project	GraphPages	GraphPage
Project	LayoutPages	LayoutPage
Project	LooseDatasetNames	string (loose datasets)
Project	MatrixPages	MatrixPage
Project	Notes	Note
Project	Page	PageBase

Project	WorksheetPages	WorksheetPage
Selection	Objects	OriginObject
TreeNode	Children	TreeNode
Worksheet	Columns	Column
Worksheet	EmbeddedPages	Page

サンプル

現在のプロジェクト内のすべてのグラフページを一覧表示

```
foreach(GraphPage gp in Project.GraphPages)
{
	out_str(gp.GetName());
}
```

現在のプロジェクト内のすべてのワークシートページを一覧表示

```
foreach(WorksheetPage wksPage in Project.WorksheetPages)
{
	out_str(wksPage.GetName());
}
```

現在のプロジェクト内のすべての行列ページを一覧表示

```
foreach(MatrixPage matPage in Project.MatrixPages)
{
	out_str(matPage.GetName());
}
```

すべてのデータプロットを一覧表示

```
GraphLayer gl = Project.ActiveLayer();
foreach(DataPlot dp in gl.DataPlots)
{
	string strRange;
```

```
        dp.GetRangeString(strRange, NTYPE_BOOKSHEET_XY_RANGE);



        printf("%d, %s", dp.GetIndex()+1, strRange);

}
```

21 索引

C

C++(.Net) と C# DLL にアクセスする 313

G

GetN ダイアログ 260
GNU Scientific Library を呼び出す 310

H

Hello World ダイアログ 289
HTML ダイアログ 288

L

LabTalk スクリプトを実行する 252
LabTalk 変数の値を取得およびセットする 251

N

NAG 関数を使用する 235

O

Origin C コードを配布する 51
Origin C コードに LabTalk スクリプトを埋め込む 253
Origin C ファイルの作成と編集 37
Origin C プログラミングガイド 1

P

Python ダイアログ 266

S

Splitter ダイアログ 279
SQLite データベースへのアクセス 248

U

Utility クラス 35

X

XY 検索 225
X ファンクション 264
X ファンクションへのアクセス 255

あ

アプリケーションコミュニケーションクラス 23

う

ウィザードダイアログ 272
ウェイトカーソル 303

え

エラーと例外処理 20

く

クラス 19
クラスの階層 337
グラフィックオブジェクトの作成とアクセス 151
グラフからデータポイントを取得 304
グラフにコントロールを追加する 305
グラフのエクスポート 195
グラフの作成と編集 120
グラフプレビュー付きダイアログ 273
グラフ付 HTML ダイアログ 295

こ

コレクション 340
コンパイル、リンク、ロード 42
コンパイルした関数を使用する 47
コンポジットデータ型のクラス 23

し

システムクラス 32

す

スクリプトウィンドウ 243

て

データプロットを追加する 126
データプロットを編集する 133
データベースからインポート 245
データベースへのエクスポート 246
データをインポートする 183
データ型と変数 7
デバッグする 46

の

ノートウィンドウ 244

ひ

ピークと基線 231

ふ

フォルダを管理する ... 172
フロー制御ステートメント ... 13
プロジェクト管理 ... 171

へ

ページにアクセスする ... 173

め

メタデータにアクセスする ... 175

ゆ

ユーザインターフェースのコントロールクラス ... 32

れ

レイヤを管理する ... 146
レポートシート ... 244

わ

ワークシート ... 90
ワークシートから行列に変換 ... 109
ワークシートデータの操作 ... 100
ワークシートのエクスポート ... 195
ワークシートの基本操作 ... 90
ワークシート列 ... 82
ワークシート列データの操作 ... 86
ワークシート列操作 ... 82
ワークブックの基本操作 ... 77
ワークブックの操作 ... 80

漢字

演算子 ... 10
仮想行列 ... 116
画像のインポート ... 189
外部 DLL から Python にアクセスする ... 324
外部アプリケーションにアクセスする ... 334
関数 ... 17
基本機能 ... 3
計算機の作成 ... 291
結果ログ ... 243
行列オブジェクトのデータ操作 ... 69
行列オブジェクトの基本操作 ... 66
行列からワークシートに変換する ... 75
行列シートのデータ操作 ... 65
行列シートの基本操作 ... 58
行列のエクスポート ... 196
行列ブックの基本操作 ... 55
信号処理 ... 229
数学 ... 199
数値データ ... 159
線形フィット ... 210
組み込みダイアログボックス ... 257
操作にアクセスする ... 180
多項式フィット ... 218
多重回帰 ... 219
単純な Hello World ダイアログ ... 266
統計 ... 208
動画のインポート ... 192
動画のエクスポート ... 197
内部 Origin オブジェクトクラス ... 25
日付と時間データ ... 169
非線形フィット ... 221
分析クラス ... 23
文字列データ ... 166