Origin C Programming Guide for Origin 8.5.1

Table of Contents

1 Basic Features

Origin C is a high level programming language closely based on the ANSI C programming language. In addition, Origin C supports a number of C++ features including classes, mid-stream variable declarations, overloaded functions, references, and default function arguments. Origin C also supports collections, and the **foreach** and **using** statements from the C# programming language.
Origin C programs are developed in Origin's Integrated Development Environment (IDE) named Code Builder. Code Builder includes a source code editor with syntax highlighting, a workspace window, compiler, linker, and a debugger. Refer to **Help: Programming: Code Builder** for more information about Code Builder.
Using Origin C allows developers to take full advantage of Origin's data import and handling, graphing, analysis, image export capabilities, and much more. Applications created with Origin C execute much faster than those created with Origin's LabTalk scripting language.

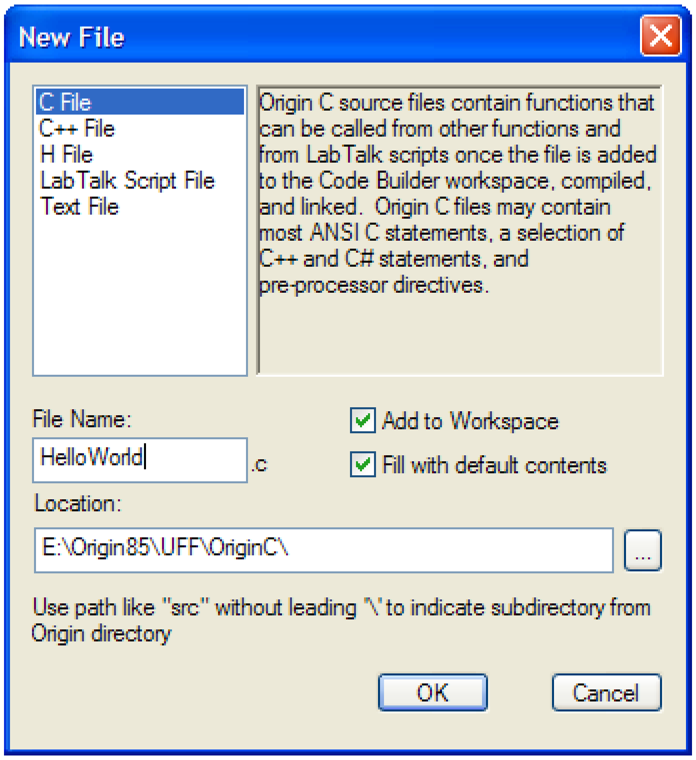
1.1 Hello World Tutorial

This tutorial will show you how to use **Code Builder** to create an Origin C function, and then access the function from Origin. Though the function itself is very simple, the steps provided here will help you get started with writing your own Origin C functions.

1. Click the **Code Builder** button on Origin's Standard toolbar to open **Code Builder**.

2. In **Code Builder**, click the **New** button on Code Builder's Standard toolbar to open the **New File** dialog.

3. Select **C File** from the list box of the dialog, and then type *HelloWorld* in the **File Name** text box.



Origin C source files contain functions that can be called from other functions and from LabTalk scripts once the file is added to the Code Builder workspace, compiled, and linked. Origin C files may contain most ANSI C statements, a selection of C++ and C# statements, and pre-processor directives.

File Name: HelloWorld .c

✓ Add to Workspace

✓ Fill with default contents

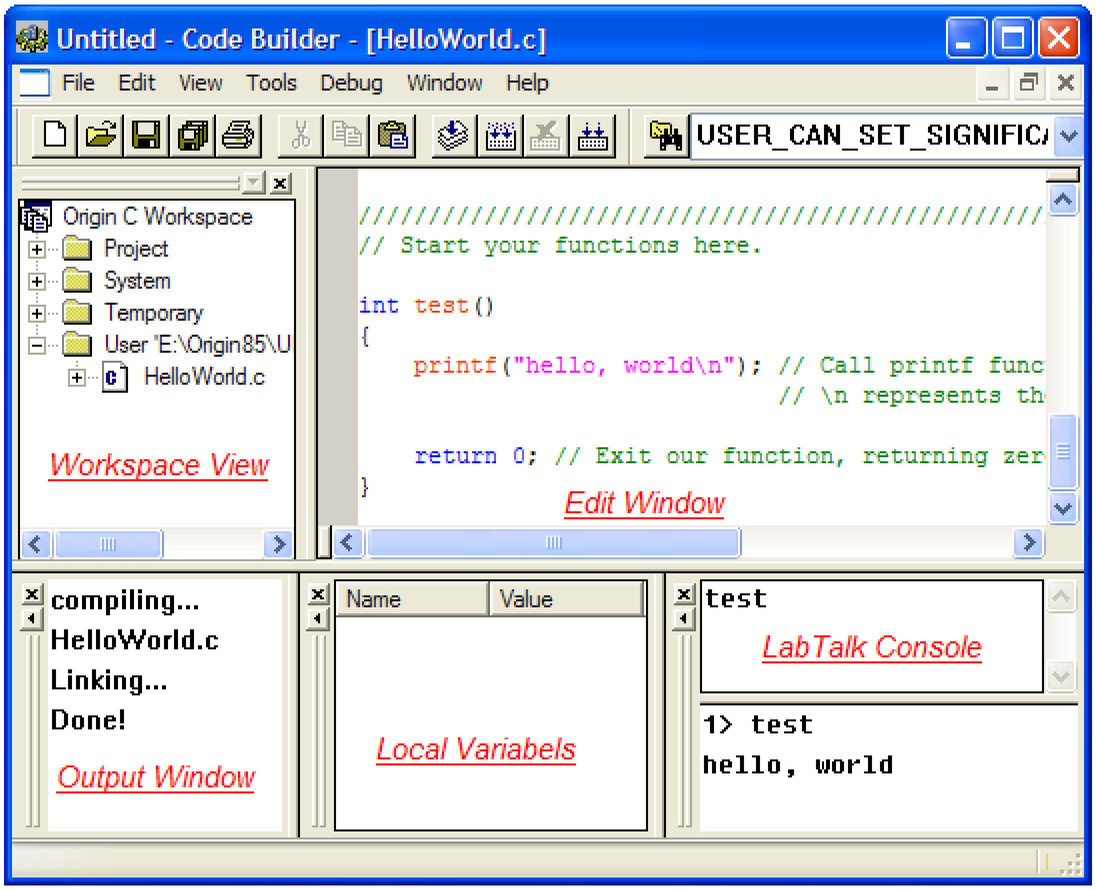
Location: E:\Origin85\UFF\OriginC\

Use path like "src" without leading '\' to indicate subdirectory from Origin directory

4. Click **OK** and the new file will be opened in Code Builder's Multiple Document Interface (MDI).
5. Copy or type the following Origin C code beneath the line that reads *// Start your functions here*.

```
int test()

{

    printf("hello, world\n"); // Call printf function to output
our text

                              // \n represents the newline
character



    return 0; // Exit our function, returning zero to the caller

}
```

6. Click the **Build** button on Code Builder's Standard toolbar to compile and link the *HelloWorld.C* source file. The **Output** window of **Code Builder** should display as



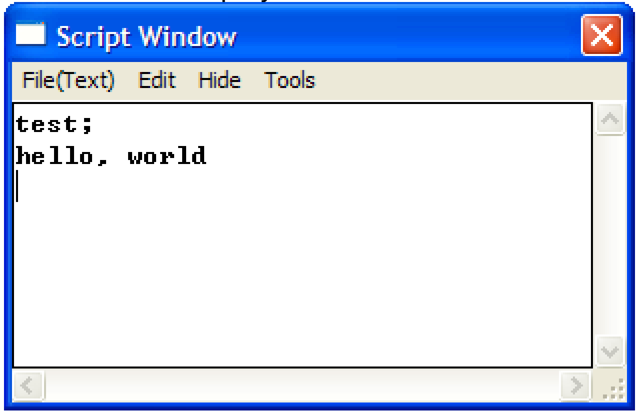
```
////////////////////////////////////////////
// Start your functions here.

int test()
{
    printf("hello, world\n"); // Call printf func
                              // \n represents th

    return 0; // Exit our function, returning zer
}
```

```
compiling...
HelloWorld.c
Linking...
Done!
```

```
test
```

```
1> test
hello, world
```

7. Now you can use this function in Origin. For example, you can call this function in Origin's **Script Window**. If the **Script Window** is not open, select the **Window: Script Window** menu item from the Origin menu to open it.
8. Type the function name *test* in the **Script Window** and then press the ENTER key to execute the command. The Origin C function will be executed and *hello, world* will be displayed in the next line.



```
test;
hello, world
```

9. Besides the **Script Window**, the function can also be called from the **LabTalk Console Window** in Code Builder. Select **View:LabTalk Console** in Code Builder if this console window is not open.

💡	Once an Origin C file has been successfully compiled and linked, all functions defined in the file can be called as script commands from anywhere in Origin that supports LabTalk script. The function

	parameters and return value need to meet certain criteria for the function to be accessible from script. To learn more, please refer to the **LabTalk Programming: LabTalk Guide: Calling X-Functions and Origin C Functions: Origin C Functions** chapter of the LabTalk help file. This help file is accessible from the **Help: Programming: LabTalk** main menu in Origin.

2 Language Fundamentals

Origin C is closely based on the ANSI C/C++ programming languages. This means Origin C supports the same data types, operators, flow control statements, user defined functions, classes and error and exception handling. The next sections will elaborate on these areas of Origin C.

2.1 Data Types and Variables

2.1.1 ANSI C Data Types

Origin C supports all the ANSI C data types: char, short, int, float, double and void. In addition, you can have an array of, and a pointer to, each of these data types.

```
char name[50];          // Declare an array of characters
unsigned char age;      // Declare an unsigned 8-bit integer
unsigned short year;    // Declare an unsigned 16-bit integer
```

2.1.2 Origin C Composite Data Types

Although the C syntax for declaring an array is supported, Origin C provides **string**, **vector** and **matrix** classes to simplify working with data types in one or two dimensional arrays. These data types include char, byte, short, word, int, uint, complex. A vector can be of type string for a string array, but a matrix cannot. A matrix can be of numerical types only.

```
string str = "hello, world\n";          // Declare and initialize a
string

vector<double> vA1 = {1.5, 1.8, 1.1}; // Declare and initialize doubles
vector vA2 = {2.5, 2.8, 2.1, 2.4};

vector<string> vs(3);                   // Declare a string array
vs[0] = "This ";                        // Assign string to each string
array item
vs[1] = "is ";
vs[2] = "test";

matrix<int> mA1;                        // Declare a matrix of integers
matrix mA2;                             // Declare a matrix of doubles

// NOTE: The double data type is implied when a data type is not
// specified in the declaration of vector and matrix variables.
```

Another useful class provided by Origin C is the **complex** class. The complex class supports numeric data containing both a real and an imaginary component.

```
complex cc(4.5, 7.8);         // Declare a complex value.
                              // The real component is set to 4.5 and
                              // the imaginary component is set to 7.8
out_complex("value = ", cc); // Output the complex value
```

2.1.3 Color

Colors in Origin C are represented with a DWORD value. These values can be an index into Origin's internal color palette or an actual color composed of red, green, and blue components.

Palette Index

Origin's internal Palette contains 24 colors. An index into Origin's internal color palette is a zero based value from 0 to 23. Origin C provides named constants for each of these colors. Each name begins with the prefix SYSCOLOR_ followed by the name of the color. The following table lists the 24 color names and their indices.

Index	Name	Index	Name
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;
```

Auto Color

There is a special color index referred to as **Auto**. When this index is used the element will be colored using the same color as its parent. Not all elements support the **Auto** index. See Origin's graphical user interface for the element to determine if the **Auto** index is supported.
The INDEX_COLOR_AUTOMATIC macro is used when the **Auto** index value is needed.

```
DWORD dwColor = INDEX_COLOR_AUTOMATIC;
```

RGB

An Origin color value can also represent an RGB value. RGB values are made up of 8-bit red, green, and blue components. These values can easily be made using the RGB macro}.

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

The values returned from the RGB macro cannot be directly used as Origin color values. You will need to use the RGB2OCOLOR macro to convert the RGB values to Origin color values.

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

If you ever need to know whether an Origin color value represents an RGB value or an index into a palette then you can use the OCOLOR_IS_RGB macro. This macro returns true if the value represents an RGB value and returns false otherwise.

```
if( OCOLOR_IS_RGB(ocolor) )
    out_str("color value represents an RGB color");
else
    out_str("color value represents a color index");
```

Once you determine that an Origin color value represents an RGB value, then you can use the GET_CRF_FROM_RGBOCOLOR macro to extract the RGB value from the Origin color value.

```
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));
}
```

2.2 Operators

Operators support the same arithmetic, logical, comparative, and bitwise operators as ANSI C. The following sections list the four types of operators and show their usage.

2.2.1 Arithmetic Operators

Operator	Purpose
*	multiplication
/	division
%	modulus (remainder)
+	addition
-	subtraction
^	exponentiate See note below.

Note: Origin C, by default, treats the caret character(^) as an exponentiate operator. This is done to be consistent with LabTalk. ANSI C uses the caret character as the exclusive OR operator. You can force Origin C to treat the caret character as the exclusive OR operator by using a special pragma statement before your code.

```
out_int("10 raised to the 3rd is ", 10^3);
#pragma xor(push, FALSE)
out_int("10 XOR 3 is ", 10^3);
```

Dividing an integer by another integer will give an integer result by default. Use the pragma statement below before codes to make Origin C compiler to treat all numeric literals as double type.

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

#pragma numlittype(push, TRUE)
out_double("3/2 is ", 3/2); // output 1.5
```

The modulus operator calculates the remainder of the left operand divided by the right operand. This operator can only be applied to integral operands.

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

2.2.2 Comparison Operators

Comparison operators evaluate to true or false with true yielding 1 and false yielding 0.

Operator	Purpose
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to
==	equal to
!=	not equal to

```
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");
```

2.2.3 Logical Operators

Logical operators evaluate to true or false with true yielding 1 and false yielding 0. The operands are evaluated from left to right. Evaluation stops when the entire expression can be determined.

Operator	Purpose
!	NOT
&&	AND

||	OR

Consider the following two examples:

```
expr1A && expr2
expr1B || expr2
```

expr2 will not be evaluated if expr1A evaluates to false or expr1B evaluates to true. This behavior is to the programmer's advantage and allows efficient code to be written. The following demonstrates the importance of ordering more clearly.

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

In the above example the entire 'if' expression will evaluate to false if ptr is equal to NULL. If ptr is NULL then it is very important that the dataValue not be compared to the upperLimit because reading the dataValue member from a NULL pointer can cause an application to end abruptly.

2.2.4 Bitwise Operators

Bitwise operators allow you to test and set individual bits. The operator treats the operands as an ordered array of bits. The operands of a bitwise operator must be of integral type.

Operator	Purpose
~	complement
<<	shift left
>>	shift right
&	AND
^	exclusive OR (XOR) See note below.
|	inclusive (normal) OR

Note: Origin C, by default, treats the caret character as an exponentiate operator. This is done to be consistent with LabTalk. ANSI C uses the caret character as the exclusive OR operator. You can force Origin C to treat the caret character as the exclusive OR operator by using a special pragma statement before your code.

```
out_int("10 raised to the 3rd is ", 10^3);
#pragma xor(push, FALSE)
out_int("10 XOR 3 is ", 10^3);
```

2.3 Statement Flow Control

Origin C supports all ANSI C flow control statements including the if, if-else, switch, for, while, do-while, goto, break and continue statements. In addition, Origin C supports the C# foreach for looping through a collection of objects.

2.3.1 The if Statement

The **if** statement is used for testing a condition and then executing a block of statements if the test results are true. The **if-else** statement is similar to the **if** statement except the **if-else** statement will execute an alternative block of statements when the test results are false.
The following are examples of **if** statements in Origin C, using different input types:

```
bool bb = true;     // boolean type
if( bb )
{
        out_str("bb is true");
}

int nn = 5;
if( nn )            // integer type, 0 = false, non-zero = true
{
        out_str("nn not 0");
}

double* pData = NULL;
if( NULL == pData ) // check if pointer is NULL
{
        out_str("Pointer pData is NULL");
}
```

The following is a simple **if-else** block in Origin C. Note that the if-block and the else-block are enclosed in separate sets of curly braces, {}.

```
if( bRet )
{
        out_str("Valid input");      // when bRet is true
}
else
{
        out_str("INVALID input");    // when bRet is false
}
```

The curly braces are optional if the block contains only one statement. This means the above code can also be written without the braces.

```
if( bRet )
        out_str("Valid input");      // when bRet is true
else
        out_str("INVALID input");    // when bRet is false
```

2.3.2 The switch Statement

The **switch** statement is used when you need to execute a different block of statements dependent on a set of mutually exclusive choices.
Cases are executed by ascending integer, starting with the number given in the integer argument to the *switch* statement. Note that the *break* command will exit the switch-block from any of the cases.

```
switch( nType ) // integer type value as condition
{
```

```
case 1:
case 2:
    out_str("Case 1 or 2");
    break;

case 3:
    out_str("Case 3");
    // no break keyword here, so fall through to case 4

case 4:
        out_str("Case 4");
        break;

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

2.3.3 The for Statement

The **for** statement is often used to execute one or more statements a fixed number of times or for stepping through an array of data wherein each element is referenced by an index.

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

2.3.4 The while Statement

The **while** and **do-while** statements execute a block of statements until a condition has been met. The **while** statement tests the condition at the beginning of the loop and the **do-while** statement tests the condition at the end of the loop.

```
int count = 0;
while( count < 10 ) // execute statements if condition is true
{
    out_int("count = ", count);
    count++;
}
int count = 0;
do
{
        out_int("count = ", count);
        count++;
} while( count < 10 ); // execute statements if condition is true
```

2.3.5 Jump Statements

Jump statements are used to unconditionally *jump* to another statement within a function. The **break**, **continue**, and **goto** statements are considered jump statements. The following examples demonstrate these jump statements.

break

```
for( int index = 0; index < 10; index++ )
{
        if( pow(index, 2) > 10 )
                break; // terminate for loop

        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; // next index

        printf("%d\n", index);
}
```

goto

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

    out_str("Skipped statement");

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

2.3.6 The foreach Statement

The **foreach** statement is used for looping through a collection of objects. The following code loops through all the pages in the project and outputs their name and type.

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

Refer to the Collections section for a list of all the Collection based classes in Origin C.

2.4 Functions

2.4.1 Global Functions

Origin C provides many global functions for performing a variety of tasks. These global functions fall into twenty-six categories.

1. Basic IO
2. Character and String Manipulation
3. COM

4. Communications
5. Curve
6. Data Conversion
7. Data Range
8. Date Time
9. File IO
10. File Management
11. Fitting
12. Image Processing
13. Import Export
14. Internal Origin Objects
15. LabTalk Interface
16. Math Functions
17. Mathematics
18. Matrix Conversion and Gridding
19. Memory Management
20. NAG
21. Signal Processing
22. Spectroscopy
23. Statistics
24. System
25. Tree
26. User Interface

Please refer to the Global Functions section on the Origin C Wiki for a complete list of functions with examples.

2.4.2 User-Defined Functions

Origin C supports user-defined functions. A user-defined function allows Origin C programmers to create functions that accept their choice of arguments and return type. Their function will then operate on those arguments to achieve their purpose. The following creates a function named *my_function* that returns a **double** value and accepts a **double** value as its only argument.

```
double my_function(double dData)
{
        dData += 10;
        return dData;
}
```

The following code snippet shows how to call the above function.

```
double d = 3.3;          // Declare 'd' as a double value
d = my_function(d);      // Call the above function
out_double("d == ", d); // Output new value of 'd'
```

Origin C functions can also be called from LabTalk.

```
d = 3.3;             // Assign 3.3 to 'd'
d = my_function(d); // Call the above function
d=;                  // Output new value of 'd'
```

2.5 Classes

Origin C supports many built-in classes, but also allows users to create their own.

2.5.1 Origin Defined Classes

Origin C comes with predefined classes for working with Origin's different data types and user interface objects. These classes will help you quickly write Origin C code to accomplish your tasks. This section will discuss the base classes to give you an overview of the capabilities these classes offer. See the next chapter, *Predefined Classes*, or the Origin C Wiki for more details and examples of Origin defined classes.

2.5.2 User Defined Classes

Origin C supports user-defined classes. A user-defined class allows Origin C programmers to create objects of their own type with methods (member functions) and data members.
The following code creates a class named **Book** with two methods, **GetName** and **SetName**.

```
class Book
{
public:
        string GetName()
        {
                return m_strName;
        }

        void SetName(LPCSTR lpcszName)
        {
                m_strName = lpcszName;
        }
private:
        string m_strName;
};
```

And below is a simple example using the class and method definitions above to declare an instance of the **Book** class, give it a name using **SetName**, and then output the name using **GetName**.

```
void test_class()
{
        Book OneBook; // Declare a Book object

        // Call public function to Set/Get name for the Book object
        OneBook.SetName("ABC");
        out_str(OneBook.GetName());
}
```

The above example is very simple. If you want to know more class features, for example, constructors and destructors, or virtual methods, please view the EasyLR.c, EasyLR.h and EasyFit.h files under the *\Samples\Origin C Examples\Programming Guide\Extending Origin C* subfolder of Origin.

2.6 Error and Exception Handling

Origin C supports C++ exception handling using the **try**, **catch**, and **throw** statements.
The try block consists of the **try** keyword followed by one or more statements enclosed in braces. Immediately after the try block is the catch handler. Origin C supports only a single catch handler that accepts an *integer* argument. After the **catch** keyword comes one or more statements enclosed in braces.

```
try
{
    LPSTR lpdest = NULL;     // NULL pointer on purpose
    strcpy(lpdest, "Test"); // copy to NULL pointer to cause error
}
catch(int nErr)
{
    out_int("Error = ", nErr);
}
```

The try-catch works by executing the statements in the try block. If an error occurs, the execution will jump to the catch block. If no error occurs then the catch block is ignored.
The throw keyword is optional and is used to trigger an error and cause execution to jump to the catch block.

```
void TryCatchThrowEx()
{
    try
    {
        DoSomeWork(4);  // pass a valid number to show success
        DoSomeWork(-1); // pass an invalid number to cause error
    }
    catch(int iErr)
    {
        printf("Error code = %d\n", iErr);
    }
}
void DoSomeWork(double num)
{
    if( num < 0 )
        throw 100; // Force error
    if( 0 == num )
        throw 101; // Force error

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

3 Predefined Classes

In this section, the predefined classes in Origin C will be described. Please see class hierarchy as a reference for more information about the relationships among Origin C built-in classes.

3.1 Analysis Class

The following classes are used to perform data analysis. For more details, please refer to the **Origin C: Origin C Reference: Classes: Analysis** chapter in the help document of OriginC.

Class	Brief Description
NLFitContext	This class provides a method for accessing the information of the fitting function, as well as the current evaluation state that is generated by implementing the fitting function in Origin C.
NLFitSession	This class is the higher level Origin class. It wraps the NLFit class with a friendly interface to aid in implementing the fitting evaluation procedure. It is the kernel of the NLFit dialog. This class is recommended for coding in Origin C, because it takes care of all the complexities that arise from the process of interfacing to Origin.

3.2 Application Communication Class

The following classes are used to enable communication between Origin and other applications. For more details, please refer to the **Origin C: Origin C Reference: Classes: Application Communication** chapter in the help document of OriginC.

Class	Brief Description
Matlab	Used to enable communication between Origin and MATLAB.

3.3 Composite Data Types Class

The following classes are composite data types classes. For more details, please refer to the **Origin C: Origin C Reference: Classes: Composite Data Types** chapter in the help document of OriginC.

Class	Brief Description
CategoricalData	A data set of CategoricalData type is an array of integers. This array is tied to an internal Origin data set of **Text** type,

	and will be allocated and sized dynamically. A data set of this type maps the text values to categories by referring to indices (1 based offset). The text values of mapping indices are stored in the data member of CategoricalMap.
CategoricalMap	A data set of CategoricalMap type is an array of text values. This array will be allocated and sized dynamically, but not tied to any internal Origin data set. This data set contains a set of unique text values, which are sorted alpha-numerically and typically referenced by the elements of the associated object of CategoricalData type.
complex	This class is used to handle number data of complex type. It contains both the Real part and Imaginary part of the complex number.
Curve	This class is derived from the curvebase and vectorbase classes, whose methods and properties it inherits. An object of Curve type can be plotted using methods defined in the GraphLayer class easily, and it is comprised of a Y data set and, typically (but not necessarily), an associated X data set. For example, a data set plotted against row numbers will not contain an associated X data set.
curvebase	This class, which is derived from the vectorbase class, from which it inherits methods and properties, is an abstract base class and is used to handle the classes of Curve type, polymorphically. So objects of curvebase type cannot be constructed, and a derived class, such as Curve, should be used instead.
Dataset	This class is derived from the vector and vectorbase classes, and it inherits their methods and properties. A Dataset is an array, which is allocated and sized dynamically. It can be tied or not tied to an internal Origin data set. By default, the Dataset is of type double, but it can also be of any basic data type, including char, byte, short, word, int, uint and complex (but not string). The syntax Dataset<*type*> can be used to construct these types of Dataset.
Matrix	This class is derived from the matrix and matrixbase classes, from which it inherits methods and properties. A Matrix (upper case M) is a two-dimensional array, which is allocated and sized dynamically, and tied to an internal Origin matrix window. The default type of a Matrix is double, but any basic data type is allowed as well, including char, byte, short, word, int, uint and complex (but not string). The syntax Matrix<*type*> is used to construct these types of Matrix.
	This class is used to access the data in the internal Origin

	matrix, while the MatrixObject class is used to control the style of the matrix. That is to say, the relationship between the MatrixObject and Matrix classes is the same as the one between the Column and Dataset classes. The data values displayed in the cells of the Origin matrix (referenced by a Matrix object) are typically referred to, in the worksheet, as Z values, whose associated X and Y values are linearly mapped to the columns and rows of the matrix, respectively.
matrix	This class is derived from the matrixbase class, from which it inherits methods and properties. A matrix (lower case m) is a two-dimensional array, which is allocated and sized dynamically, and is not tied to any internal Origin matrix window, which provides more flexibility. The default type of a matrix is double, but any basic data type can be used as well, including char, byte, short, word, int, uint and complex (but not string). The syntax matrix<*type*> is used to construct these types of matrix.
matrixbase	This class is an abstract base class for handling the matrix and Matrix class types polymorphically. Thus, objects of matrixbase type cannot be constructed, and objects of its derived classes, such as matrix and Matrix, should be used instead.
PropertyNode	This class is only used for including the properties of different data types, such as Bool, int, float, double, string, vector, matrix, and picture, etc.
string	This class is used to construct a null terminated array of characters, which is similar to an MFC CString object. A lot of methods for manipulating strings (text data) are defined in this class. It can also be used together with the vector class by syntax vector<string> to define string arrays.
Tree	This class is used to save Origin C trees as XML files, as well as to load XML files to Origin C trees.
TreeNode	This class provides several methods for constructing multilevel trees, traversing trees and accessing the attributes of tree nodes.
TreeNodeCollection	This class is used to get a collection of child tree nodes with an enumerative name prefix.
vectorbase	This class is an abstract base class used for handling objects of vector and Dataset types polymorphically. Thus, objects of

	this type cannot be constructed, and objects of its derived classes, such as vector and Dataset, should be used instead.
vector	This class is derived from the vectorbase class, from which it inherits methods and properties. A vector is an array, which is allocated and sized dynamically, and not tied to any internal Origin data set, which allows for more flexibility. The default type of vector is double, but other basic data types are also allowed, including char, byte, short, word, int, uint, complex, and string. The syntax vector<*type*> can be used to construct these types of vector.

3.4 Internal Origin Objects Class

The following classes are used to handle Origin objects. For more details, please refer to the **Origin C: Origin C Reference: Classes: Internal Origin Objects** chapter in the help document of OriginC.

Class	Brief Description
Axis	This class is derived from the OriginObject class, and can be used to access Origin axes. Origin axes are contained by layers on an Origin page.
AxisObject	This class is derived from the OriginObject class, and can be used to access Origin axis objects, including axis ticks, grids and labels. Origin axis objects are contained by axes on an Origin graph page.
Collection	This class provides a template for collections of various internal Origin objects, such as Pages (the collection of all PageBase objects in a project file), etc. This class contains an implicit templatized type _TemplType, which is the type of one element of the collection. For example, the templatized type of the Pages collection in the Project class (Collection<PageBase> Pages;) is PageBase. Each collection usually has a parent class, whose data member is the collection. For example, Collection<PageBase> Pages is one member of the Project class, because Project contains all the pages. Therefore, each collection can be attached or unattached to an internal object. All collections can use the methods defined in the Collection class. The foreach loop is the most useful way for looping once for each of the elements in the collection.

CollectionEmbeddedPages	This class is used to access the pages embedded in a worksheet.
Column	This class is derived from the DataObject, DataObjectBase and OriginObject classes, and it inherits their methods and properties. In this class, methods and properties are provided for dealing with Origin worksheet columns. A worksheet object contains a collection of Column objects, and each Column object holds a Dataset. A Column object is mainly used for controlling the style of data in the associated Dataset. A Column object is a wrapper object, which refers to an internal Origin column object, but does not actually exist in Origin.
DataObject	This class is derived from the DataObjectBase class, and is the base class of worksheet columns and matrix objects. Origin data objects are contained in layers on an Origin page. For example, columns (data objects) are contained in a worksheet (layer) on a worksheet window (page).
DataObjectBase	This class is an abstract base class, which provides methods and properties for handling the class types related to DataObject and DataPlot, polymorphically. Thus, objects of this type cannot be constructed, and objects of its derived classes, such as DataObject, Column, MatrixObject and DataPlot, should be used instead.
DataPlot	This class is derived from the DataObjectBase and OriginObject classes, from which it inherits methods and properties. In this class, methods and properties are provided for Origin data plots. An internal Origin data plot object is used to store the characteristics of the Origin data plot, and it is contained in a graph layer on a graph page. A DataPlot object is a wrapper object, which refers to an internal Origin data plot object and does not actually exist in Origin. Thus, multiple wrapper objects can refer to the same internal Origin object.
DataRange	Methods and properties are provided in this class for constructing data ranges and accessing data in a Worksheet, Matrix or Graph window. This class does not hold data by itself, it just keeps the data range with the page name, sheet name (layer index for a graph) and row/column indices (data plot indices for a

	graph). Multiple data ranges can be contained in one DataRange object, and the sub data range can be the whole data sheet, one column, one row, multiple continuous columns, or multiple continuous rows.
DataRangeEx	This class is the extensional class of DataRange.
DatasetObject	This class is used to access non-numeric data sets, which are usually members of Column objects.
Datasheet	This class is derived from the Layer and OriginObject classes, and it inherits their methods and properties. This class is used to handle Origin worksheet and matrix layers.
Folder	Project Explorer is a user interface inside Origin with folder/sub-folder structure, just likes Window Explorer. It is used to organize and access graph, layout, matrix, note, and worksheet windows in an Origin project file. The Folder class has the ability to access the methods and properties of Project Explorer, and contains collections of all Origin pages and Project Explorer folders. A Folder object is a wrapper object, which refers to an internal Origin Project Explorer object but does not actually exist in Origin. Thus, multiple wrapper objects can refer to the same internal Origin object.
fpoint3d	This class is used to handle data points that are located in three-dimensional space, with double type for their (x, y, z) coordinates.
fpoint	This class is used to handle data points that are located in two-dimensional, or planar, space and use double type for their (x, y) coordinates.
GetGraphPoints	This class is used to get the position (x, y) of a screen point or data point from an Origin graph window.
GraphLayer	This class is derived from the Layer and OriginObject classes, and it inherits their methods and properties. In this class, methods and properties are provided for Origin graph layers. Internal Origin graph pages contain one or more graph layers, and graph layers contain one or more data plots. Thus, the GraphPage class contains a

	collection of GraphLayer objects, and the GraphLayer class contains a collection of DataPlot objects. A GraphLayer object is a wrapper object, which refers to an internal Origin graph layer object, but does not actually exist in Origin. So multiple wrapper objects can refer to the same internal Origin object.
GraphObject	This class is derived from the OriginObject class, from which it inherits methods and properties. In this class, methods and properties are provided for handling Origin graph objects, which include text annotations, graphic annotations (e.g. rectangles, arrows, line objects, etc.), data plot style holders, and region of interest objects. Origin graph objects are generally contained in layers on an Origin page, thus the GraphLayer class contains a collection of GraphObjects. A Graph object is a wrapper object, which refers to an internal Origin graph object and does not exist in Origin. So multiple wrapper objects can refer to the same internal Origin object.
GraphPage	This class is derived from the Page, PageBase, and OriginObject classes, and it inherits their methods and properties. In this class, methods and properties are provided for handling internal Origin graph pages (windows). A GraphPage object is a wrapper object, which refers to an internal Origin graph page object but does not exist in Origin. Thus, multiple wrapper objects can refer to the same internal Origin object. The Project class contains a collection of GraphPage objects, named GraphPages, in the open project file. A GraphPage object can be used to locate and access layers on an Origin graph page, which can then be used to access objects in the layer, such as DataPlots or GraphicObjects.
GraphPageBase	This class is the base class for GraphPage and LayoutPage.
Grid	This class is used to set the format of data sheet windows (Origin worksheets and matrix sheets). Extra functions are also provided in this class for data selection, showing column/row labels, setting cell text color, merging cells, and so on.
GroupPlot	This class is derived from the OriginObject class and can be used to handle Origin group plots. GroupPlot objects are contained in layers on an Origin page.

Layer	This class is derived from the OriginObject class, from which it inherits methods and properties. In this class, methods and properties are provided for handling internal Origin layers. All Origin pages (windows), except note pages, contain one or more layers. Origin objects found "on" a page are generally contained by layers which are themselves contained by the page. Many graph objects are contained in layers, thus the Layer class contains the collection of graph objects. A Layer object is a wrapper object, which refers to an internal Origin layer object but does not actually exist in Origin. So multiple wrapper objects can refer to the same internal Origin object.
LayoutPage	This class is derived from the Page, PageBase, and OriginObject classes, from which it inherits methods and properties. In this class, methods and properties are provided for handling internal Origin layout pages (windows). The Project class contains a collection of LayoutPage objects. A LayoutPage object is a wrapper object, which refers to an internal Origin layout page object and does not exist in Origin. So multiple wrapper objects can refer to the same internal Origin object.
Layout	This class is derived from the Layer and OriginObject classes, and it inherits their methods and properties. In this class, methods and properties are provided for handling internal Origin layout layers. Origin layout pages contain a layout layer, which contains other objects. A Layout object is a wrapper object, which refers to an internal Origin layout object but does not exist in Origin. So multiple wrapper objects can refer to the same internal Origin object.
MatrixLayer	This class is derived from the Datasheet, Layer, and OriginObject classes, from which it inherits methods and properties. In this class, methods and properties are provided for handling matrix layers in Origin matrix pages. An Origin matrix contains a number of matrix objects, thus the MatrixLayer class contains a collection of the matrix objects in the matrix layer. A MatrixLayer object is a wrapper object, which refers to an internal Origin matrix layer object, and does not actually exist in Origin. So multiple wrapper objects can refer to the same internal Origin object.

MatrixObject	This class is derived from the DataObject, DataObjectBase, and OriginObject classes, and it inherits their methods and properties. This class is used to handle internal Origin matrix objects. MatrixObject is mainly used to control the style of the data in the internal Origin matrix, while the Matrix class is used to access the data in the matrix. Thus, the MatrixObject class has the same relationship with the Matrix class as the Column class has with the Dataset class. That is to say, an internal Origin matrix object (MatrixObject) holds a matrix data set (Matrix), just like a worksheet column (Column) holds a data set (Dataset). The data values displayed in the cells of a matrix are considered Z values, whose associated X and Y values are linearly mapped to the columns and rows of the matrix, respectively. A MatrixLayer holds a collection of MatrixObjects, even though there is generally only one MatrixObject per MatrixLayer. A MatrixObject is a wrapper object, which refers to an internal Origin matrix object yet does not actually exist in Origin . So multiple wrapper objects can refer to the same internal Origin object.
MatrixPage	This class is derived from the Page, PageBase, and MatrixPage classes, from which it inherits methods and properties. In this class, methods and properties are provided for handling internal Origin matrix pages (windows). A MatrixPage object is a wrapper object, which refers to an internal Origin matrix page object but does not exist in Origin. So multiple wrapper objects can refer to the same internal Origin object. The Project class contains a collection of MatrixPage objects, named MatrixPages, in the open project file. A MatrixPage object can be used to locate and access layers on the Origin matrix page, which can then be used to access objects in the layers, such as MatrixObjects and GraphicObjects.
Note	This class is derived from the PageBase and OriginObject classes, from which it inherits their methods and properties. In this class, methods and properties are provided for handling internal Origin Note pages (windows). The Project class contains a collection of Note objects. A Note object is a wrapper object, which refers to an internal Origin Note page but does not actually exist in Origin. And so, multiple wrapper objects can refer to

	the same internal Origin object.
OriginObject	This class is the Origin C base class for all Origin objects. Member functions and data members are provided in this class for all Origin objects.
Page	This class is derived from the PageBase and OriginObject classes, and it inherits their methods and properties. In this class, methods and properties are provided for handling internal Origin pages, which contain one or more layers (except Note windows). The Page class contains a collection of the layers in the page. A Page object is a wrapper object, which refers to an internal Origin page object but does not exist in Origin. So multiple wrapper objects can refer to the same internal Origin object.
PageBase	This class provides methods and properties for internal Origin pages (windows). Usually, this class is used in one of two ways. One way is by using a PageBase object as a parameter of a general function, but not using a specific Page object. The other way is by attaching a PageBase object to an unknown active page. Both usages can handle the specific page objects polymorphically. That is also the purpose of this class: to act as an abstract class for its derived page types, which include Note, GraphPage, WorksheetPage, LayoutPage, and MatrixPage.
point	This class is used to handle data points located in two-dimensional, or planar, space, with integer (x, y) coordinates.
Project	This class provides methods and properties for accessing most objects in an Origin project file. The Project class includes collections of different page types, and collections of all the data sets (including loose data sets, that are not in a worksheet column) in the Project file. This class also provides methods for getting active objects in a project file, as well as RootFolder properties, including ActiveCurve, ActiveLayer, and ActiveFolder. A Project object is a wrapper object, which refers to an internal Origin project object but does not actually exist in Origin. Only one project file can be open in Origin at a time, so all Project objects refer to the currently open project file.

ROIObject	This class is derived from the GraphObject class, from which it inherits methods and properties. In this class, methods and properties are provided for working with Origin region of interest objects. An Origin region of interest object is used to identify a region of interest in an Origin matrix. A ROIObject is a wrapper object, which refers to an internal Origin region of interest object but does not actually exist in Origin. So multiple wrapper objects can refer to the same internal Origin object.
Scale	This class is derived from the OriginObject class, from which it inherits methods and properties. In this class, methods and properties are provided for handling Origin axis scales. Two scale objects (X scale and Y scale) are contained in every graph layer on a graph page. A Scale object is a wrapper object, which refers to an internal Origin scale object but does not actually exist in Origin. This means that multiple wrapper objects can refer to the same internal Origin object.
storage	Origin allows for saving binary type (TreeNode type) and INI type (INIFile type) information in Origin objects, which can be any Origin C objects derived from the OriginObject class, such as a WorksheetPage, Column, Folder, GraphPage, GraphLayer, DataPlot, Project, etc.
StyleHolder	This class is derived from the GraphObject and OriginObject classes, and it inherits their methods and properties. In this class, methods and properties are provided for data plot style holders. A data plot style holder is used to store plot type information. A StyleHolder object is a wrapper object, which refers to an internal Origin StyleHolder object but does not actually exist in Origin. So multiple wrapper objects can refer to the same internal Origin object.
UndoBlock	This class provides two functions for accessing projects safely, UndoBlockBegin() and UndoBlockEnd().
WorksheetPage	This class is derived from the Page, PageBase, and OriginObject classes, and it inherits their methods and properties. In this class, methods and properties are provided for internal Origin worksheet pages (windows). The Project class contains a collection of

	WorksheetPage objects. A WorksheetPage object is a wrapper object, which refers to an internal Origin worksheet page object, but does not actually exist in Origin. So multiple wrapper objects can refer to the same internal Origin object.
Worksheet	This class is derived from the Datasheet, Layer, and OriginObject classes, from which it inherits methods and properties. In this class, methods and properties are provided for handling worksheet layers on Origin worksheet pages. An Origin worksheet may contain a number of worksheet columns, thus the Worksheet class contains a collection of all the columns in the worksheet. A Worksheet object is a wrapper object, which refers to an internal Origin worksheet object, and does not exist in Origin. So multiple wrapper objects can refer to the same internal Origin object.
XYRange	This class is derived from the DataRange class, from which it inherits methods and properties. By using methods defined in this class, the data range, which has one independent variable (X) and one dependent variable (Y), can be gotten from matrix and worksheet windows, and put into matrix and worksheet windows. It can also be used to make a plot on a graph window. Just like the DataRange class, XYRange does not hold data itself, but just keeps the data range with page name, sheet name (layer index for a graph) and row/column indices (data plot indices for a graph). Every XYRange object can contain multiple sub XY data ranges.
XYRangeComplex	This class is derived from the XYRange and DataRange classes, and it inherits their methods and properties. This class is used to get and set XY data sets of complex type for matrix and worksheet windows. Just like the DataRange class, the XYRangeComplex class does not hold data itself, but just keeps the data range with page name, sheet name and row/column indices. Every XYRangeComplex object can contain multiple sub XY complex data ranges.
XYZRange	This class is derived from the DataRange class, from which it inherits methods and properties. This class is used to get and set XYZ data sets for matrix and worksheet windows. Just like the DataRange class, the XYZRange class

	does not hold data itself, but just keeps the data range with page name, sheet name and row/column indices. Every XYZRange object can contain multiple sub XYZ data ranges.

3.5 System Class

The following classes are about system settings. For more details, please refer to the **Origin C: Origin C Reference: Classes: System** chapter in the help document of OriginC.

Class	Brief Description
file	This class is used to control the permission to read/write the binary files by using unbuffered io (accessing immediate disk). It is similar to the MFC CFile class. Please also refer to the stdioFile class, which is for buffered stream io to text files.
INIFile	This class is used to access the data stored in the initialization file.
Registry	The methods in this class are used to access Windows registry.
stdioFile	This class is derived from the file class, from which it inherits methods and properties. This class is used to control the permission to read/write the text and binary files by using buffered stream io. However, this class does not support stream io to stdin, stdout, and stderr. Please also refer to the file class, which is for unbuffered io to binary files.

3.6 User Interface Controls Class

The following classes are about user interface. For more details, please refer to the **Origin C: Origin C Reference: Classes: User Interface Controls** chapter in the help document of OriginC.
The classes marked with * are only available in Origin with the DeveloperKit installed.

Class	Brief Description
*BitmapRadioButton	This class provides the functionality of bitmap radio button controls.
*Button	This class provides the functionality of button controls. A button control is a small rectangular child window, which can be clicked on and off. The button will change its appearance when clicked. Typical buttons include check

	boxes, radio buttons and push buttons.
*CmdTarget	This class is the base class for message map architecture. A message map is used to send a command or message to the member functions you have written, and then the member functions handle the command or message. (A command is a message from a menu item, command button, or accelerator key.) Two key framework classes are derived from this class: Window and ObjectCmdTarget. To create a new class for handling messages, you can just derive your new class from one of these two classes. There is no need to derive from CmdTarget directly.
*CodeEdit	This class is derived from the RichEdit class. It is used to display the redefined color for key words in coding text.
*ColorText	This class is only available in Origin packages that have the DeveloperKit installed.
*ComboBox	This class is used to define combobox control.
*Control	This class provides the base functionality of all controls.
*DeviceContext	This class is used to define device-context objects.
*Dialog	This class is the base class for displaying dialog boxes on the screen.
*DialogBar	This class is used to create a Dockable control bar with a child Origin C-driven dialog.
*DynaControl	This class is used to generate various types of customized interface controls dynamically, such as an edit box, combo box, check box, or radio button. The values will be stored in a tree node, and the on dialog will display as a tree structure.
*Edit	This class is used to create edit controls. An edit control is a rectangular child window, which can be filled with text.
*GraphControl	This class is derived from the OriginControls, Control and Window classes, from which it inherits methods and properties. Methods defined in this class can be used to display an Origin Graph within the specified control on the dialog.

GraphObjTool	This class is the base class of GraphObjCurveTool. It is used to create and manage a rectangle on an Origin graph window, around the region of interest and containing the data.
GraphObjCurveTool	This class is derived from GraphObjTool, from which it inherits methods and properties. With these methods and properties, it can be used to create and manage a rectangle on an Origin graph window, around the region of interest and containing the data. This class also provides methods for adding a context menu and the related event functions.
*ListBox	This class is used to define list boxes. A list box shows a list of string items for viewing and selecting.
*Menu	This class is used to handle menus, including creating, tracking, updating and destroying them.
*OriginControls	This class is the base class for displaying the Origin window on dialog.
*PictureControl	This class is used to paint a PictureHolder object within the control on dialog.
progressBox	This class provides methods and properties for opening and controlling progress dialog boxes. A progress dialog box is a small dialog box that indicates the software is busy processing data. This dialog box contains a progress bar for showing the fraction of the completed processing. The progress dialog box is usually used in iterative loops.
*PropertyPage	This class is used to construct individual page objects of property sheets in a wizard dialog.
*PropertySheet	This class is used to construct property sheets in a wizard dialog. One property sheet object can contain multiple property page objects.
*RichEdit	This class provides methods for formatting text. A *rich edit control* is a window, in which text can be written and edited. The text can be in character and paragraph formatting.
*Slider	A *slider control* is a window with a slider and optional ticks. When the slider is moved by the mouse or the directional keys on the keyboard, the control will send a notification message to implement the change.

*SpinButton	A *spin button control* is a pair of arrow buttons that can be used to increase or decrease a value, such as scroll position or the number displaying in an accompanying control. This value is called the current position.
*TabControl	A tab control is used to display different information under different tabs in a dialog. This class provides methods to add/delete tab items for displaying a group of controls.
waitCursor	A wait cursor is a visual sign for indicating that the software is busy processing data. This class provides methods and properties for opening and controlling wait cursors.
*Window	This class is the base class of all window classes. It is similar to the MFC CWnd class.
*WizardControl	This class is used to construct wizard controls for implementing something step by step in a dialog. The methods available in this class enable you to add/delete steps.
*WizardSheet	This class is used to construct property sheet objects in a wizard dialog. A property sheet contains one or more property page objects.
*WorksheetControl	This class is derived from the OriginControls, Control and Window classes, and it inherits their methods and properties. The methods available in this class can be used to display an Origin Worksheet within the specified control in a dialog.
*WndContainer	This class is the base class of the derived control classes.

3.7 Utility Class

For more details about the following classes, please refer to the **Origin C: Origin C Reference: Classes: Utility** chapter in the help document of OriginC.

Class	Brief Description
Array	This class is a collection of almost all data types and objects. When Array::IsOwner is TRUE, the array will be the owner of the memories that are allocated to the objects. And the objects will be destroyed when the array is resized or destructed.
BitsHex	This class is used to compress byte vectors (1 and 0) to hexadecimal

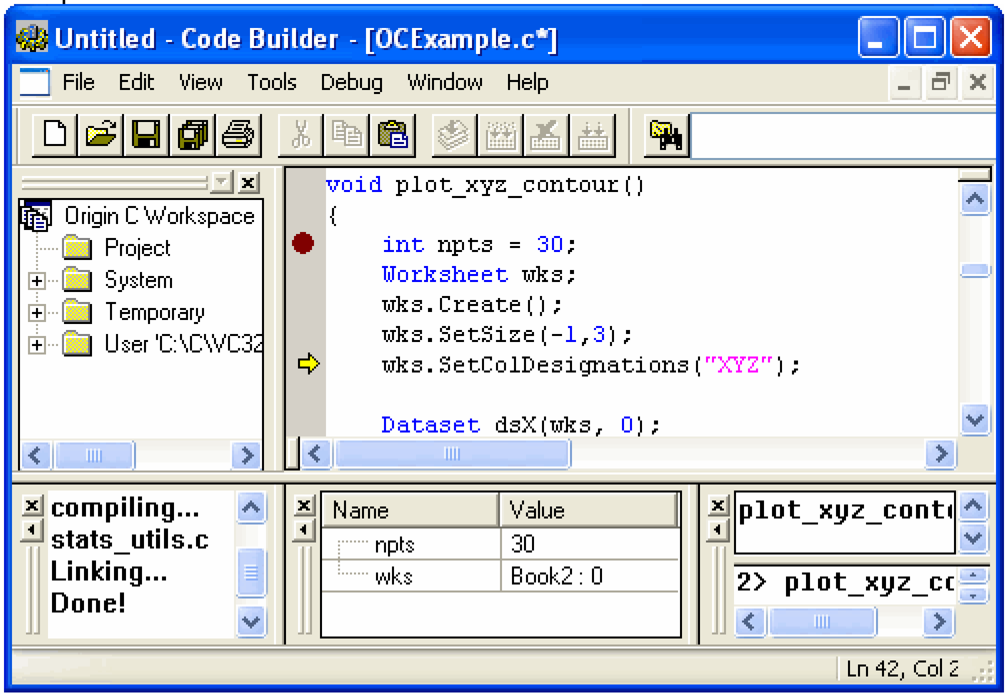
	strings, and decompress hexadecimal strings to byte vectors.
Profiler	This class can be used to measure the call times of various functions to find out the slower ones.

4 Creating and Using Origin C Code

4.1 Create and Edit an Origin C File

4.1.1 Overview

Code Builder is an Integrated Development Environment (IDE) for Origin C and LabTalk programming. Code Builder provides tools for writing/editing, compiling, linking, debugging, and executing your Origin C code. Although Origin C code can be written in any text editor, it must be added to Code Builder's Workspace to be compiled and linked.



```
void plot_xyz_contour()
{
    int npts = 30;
    Worksheet wks;
    wks.Create();
    wks.SetSize(-1,3);
    wks.SetColDesignations("XYZ");

    Dataset dsX(wks, 0);
```

```
compiling...
stats_utils.c
Linking...
Done!
```

Name	Value
npts	30
wks	Book2 : 0

```
plot_xyz_cont
2> plot_xyz_c
```

The Code Builder window

4.1.2 File Types

Origin C utilizes four types of files: source, object, preprocessed, and workspace.

Source (*.c, *.cpp, *.h)

Source files are essentially text files that contain human-readable Origin C code. You may create them in Code Builder or another text editor, and save them to any location. Code Builder's text editor provides syntax coloring, context-sensitive help and debugging features.
Until source files have been compiled, linked and loaded, the functions they contain cannot be used in Origin.

Object (*.ocb)

When a source file is compiled, an object file is produced. The object file will have the same file name as the source file, but will be given the ***.ocb** file extension. The object file is machine readable, and is what Origin uses to execute functions that are called. Origin compiles a source file for the first time, and then recompiles only when the source file is changed.
Object files in Origin are version specific, and therefore, sharing them is discouraged. If you wish to share some functions or Origin C applications, share preprocessed files instead.

Preprocessed (*.op)

By default, Origin compiles source files to produce object files. However, the system variables below can be changed to produce a preprocessed file instead of an object file. Preprocessed files still require compiling, but have the following advantages for code sharing:

- Origin version independent
- Functions can be shared without sharing source code
- The build process happens much faster than with source files

The system variables that allow you to produce either object (OCB) or preprocessed (OP) files are @OCS, @OCSB, and @OCSE. You can change their values in the Script Window or in the Code Builder LabTalk Console. For example, in the Script Window, enter:

```
@OCSB=0; // Hereafter, on compile, generate OP files
```

@OCS

The default value of this variable is 1, which allows you to create an OCB file or OP file. If @OCS=0, the compiler will not create an OCB file or an OP file.

@OCSB

The default value of @OCSB=1; this generates an object file at compile time. To generate an OP file, set @OCSB=0, after which OP files will be generated at compile time. The OP file will be saved in the same folder as its source file and have the same file name, but with the OP extension. Note that if @OCS=0, this variable is meaningless.

@OCSE

This variable is only available in OriginPro. Its default value is 0. After setting @OCSE=1, the compiler will generate an encrypted OP file, which will hide implementation details. The encrypted OP file can be loaded and linked by either Origin or OriginPro versions. **Note**: this variable is only meaningful when @OCSB=0.

Workspace (*.ocw)

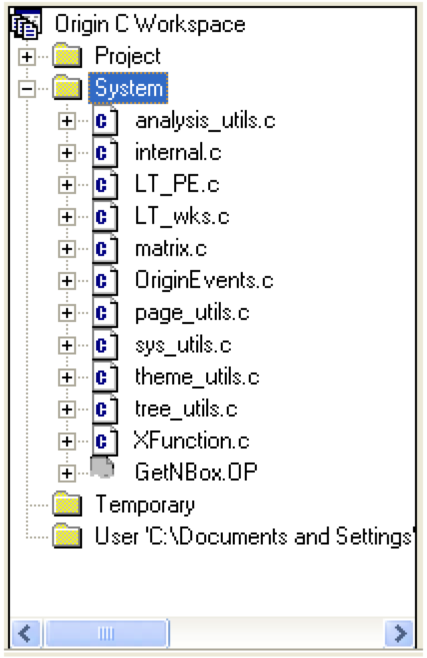
In Code Builder, you may create or use a project that contains many Origin C source files. These files may or may not be hierarchically organized in folders. It would be very inconvenient to have to load many such files manually each time you switched between projects.

For this reason, the structure and files contained in the User folder can be saved to a workspace file. Upon loading a workspace file into Code Builder, a project is restored to the state in which it was last saved; all of your source files are available in whatever structure they were assigned.

4.1.3 The Workspace View

The Code Builder Workspace view contains four folders:

1. Project
2. System
3. Temporary
4. User



```
Origin C Workspace
  Project
  System
    analysis_utils.c
    internal.c
    LT_PE.c
    LT_wks.c
    matrix.c
    OriginEvents.c
    page_utils.c
    sys_utils.c
    theme_utils.c
    tree_utils.c
    XFunction.c
    GetNBox.OP
  Temporary
  User 'C:\Documents and Settings'
```

The Workspace View

The files in each folder are compiled and linked following different events.

<u>Project</u>

Files in the Project folder are saved within the current Origin project file (*.OPJ). They are added to the Project folder of the Code Builder workspace when you open an Origin project file containing them. They are automatically compiled and linked upon opening the project file.

<u>System</u>

Files in the System folder are externally saved in Windows folders (usually in the Origin C folder or one of its subfolders). They are automatically added to the System folder of the Code Builder workspace, compiled, and linked whenever Origin starts.

<u>Temporary</u>

All files that are not listed in the Project, System, or User folders, and get loaded and compiled when using Origin, will appear in the Temporary folder. For example, if you export a graph then all the files used for handling a graph export will appear in the Temporary folder.

User

Files in the User folder are externally saved in Windows folders and are manually added to the User folder of the Code Builder workspace, compiled, and linked by the user in Code Builder.

4.1.4 Code Builder Quick Start

Get started using Code Builder in just a few steps:

1. Open Code Builder by pressing Alt+4 on the keyboard or by clicking the Code Builder toolbar button .
2. Create a new source code file by pressing Ctrl-N or by clicking the New toolbar button. When the New File dialog appears enter a name for your source code file and then press Enter or click the OK button.
3. An editor window will open. Go to the end of the last line in the editor window and press enter to start a new blank line. Enter the following function:

```
void HelloWorld()

{

        printf("Hello World, from Origin C\n");

}
```

4. Before we can call this function we need to compile and link the code. You can do this by pressing Shift+F8 or by clicking the Build toolbar button .
5. The Output window will show the compiling and linking progress. If any errors appear, then double check your function and fix the errors. When no errors appear, the function is ready to be called.
6. Click in the top part of the Command & Results window. Type the name of your function and press Enter. In the bottom part of the Command & Results window you should see a repeat of your function's name, and the line you entered, followed by a line with your function's output.

While these steps are sufficient to get you going with Code Builder, there are many more details that will help you write, debug and execute your Origin C files effectively. These are covered in the sections that follow.

4.2 Compiling, Linking and Loading

Before you can access your Origin C functions, you will need to compile and link them (a process known as building) using Code Builder.
Once your functions compile and link without error, they are loaded automatically, and you will be able to access them from the current Origin session. To access your functions in future sessions of Origin you will need to ensure that they are reloaded and linked; a process that is fast and can be automated.

This chapter covers the manual and automated build process for Origin C source files and preprocessed files.

4.2.1 Compiling and Linking

In order to make the functions defined in an Origin C source file or preprocessed file executable for the first time, the following steps are necessary:

- Add the file to the Code Builder workspace
- Compile the file
- Link the file to all dependents, compiling dependents where necessary, and load the object files that are created.

The act of compiling and linking all the files is referred to as building.

Add the File to the Workspace

Before a source file or preprocessed file can be compiled and linked, the file must be added to one of the **Code Builder** workspace folders: Project, User, System, or Temporary. Note that all source files are initially created or loaded into the User folder.

Compile the File

After adding the file to the workspace, it needs to be compiled (by clicking the **Compile** button) to generate the object file, which will have the same name as the source/preprocessed file, but with the **OCB** file extension. In Origin versions 8.1 and later, the object file will be saved in the **Application Data** folder. In older versions the file was saved to the **User Files\OCTemp** folder.

Build the Workspace

To build the active file and all its dependents, select the **Build** button, or select the **Rebuild All** button to build all files in the workspace. The object file that is created will be automatically loaded into memory and linked so that the functions defined in the file are executable within Origin.
Once the object file is generated, subsequent build processes will be much faster. If there are no changes to the built source/preprocessed file, **Code Builder** will load and link the object file directly, but not rebuild the file.

Build vs. Build All

Build: All of the files in a given folder are compiled and linked when that folder is the active window in the Code Builder (or a dependent of the active window) and the Build toolbar button is clicked.
Build All: files in ***all*** Code Builder folders are compiled and linked when the Code Builder Rebuild All toolbar button is clicked.

4.2.2 Automated Building

Initially, all Origin C source or preprocessed files are created or opened in the User folder, and the discussion above gives details for manually building Origin C source files. Many times, however, it is advantageous to automate the build process.

This can be done by making use of Code Builder's folder structure, each with slightly different functionality, or by utilizing the **Build on Startup** option:

- Add files to Code Builder's **Project folder**, and they will be built automatically each time the associated Origin project is opened.
- Add files to Code Builder's **System folder**, and they will be built automatically each time Origin starts. Files in the System folder are also rebuilt each time an Origin project or a Code Builder project is started or opened.
- The **Build on Startup** option will build the most recently opened Code Builder workspace upon Origin startup.

Add files to the Project Folder

You can add files to the System folder using the following methods:

- Right-click on the Project folder and choose **Add files**.
- Drag a file from another Workspace folder and drop it on the Project folder.

Add files to the System Folder

You can add files to the System folder using the following methods:

- Right-click on the System folder and choose **Add files**.
- Drag a file from another Workspace folder and drop it on the System folder.
- Edit the **OriginCSystem** section in the **Origin.ini** file. Editing this section must be done when Origin is not running. The section contains an entry for each file.

Build Workspace on Origin Startup

When Origin starts it will examine the contents of the Origin C Workspace System folder and if it finds any changed files then it will try to compile and link them. You also can have this procedure done to the files in the User folders by enabling the Build on Startup option.

- Activate Code Builder
- If the Workspace view is not visible then choose **Workspace** on the **View** menu.
- Right-click on **Origin C Workspace**.
- If the **Build on Startup** item is not checked then click it.

The next time you start Origin it will check the files in the User folder and try to compile and link any changed files.

Build Individual Source File on Origin Startup

The following steps demonstrate how to modify Origin.ini to load and compile Origin C source file on startup.

1. Keep Origin is close. Open Origin.ini file. This file under User File Folder(run %Y= in Command window to see current User File Folder), if cannot find it, turn to Origin install folder.
2. In [Config] section, uncomment (remove ;) OgsN = OEvents. N here can be any number just when this number not be used. Save and close this file after change.
3. Open OEvents.ogs under Origin Install folder, go to [AfterCompileSystem] section, add the following line as a new line

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

4. Save and close this file.

5. Restart Origin, open Code Build, in Temporary folder, there are 3 files. AscImpOptions depend on fu_utils.c and Import_utils.c, so compiler compile it with the two files together. More details please search run.LoadOC in Labtalk Help.

4.2.3 Building by Script

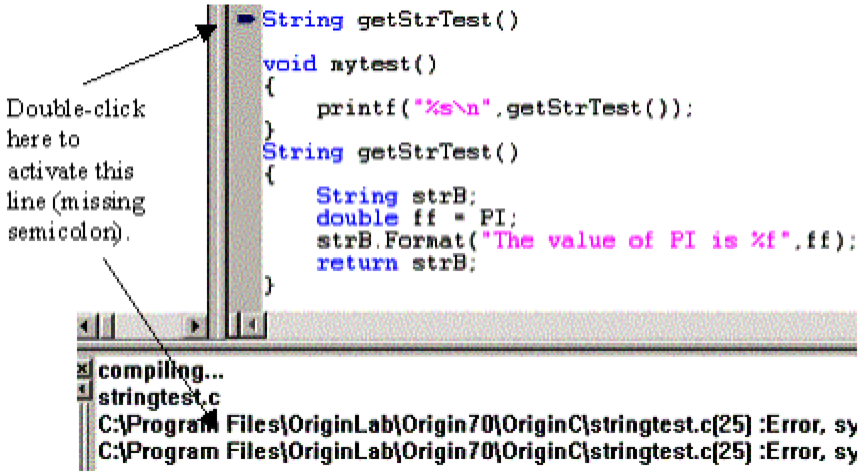
When you want to call an Origin C function in LabTalk script, you need to make sure the source file has been compiled and linking is done. You can then use the LabTalk command *Run.LoadOC* to compile and link the specific source file. For example:

1. Choose File->New Workspace... to create a new workspace. The Temporary folder should be empty now.
2. Run the following script in the Command Window... the *dragNdrop.c* file together with its dependent files all are loaded into the Temporary folder and compiled.

```
if(run.LoadOC(OriginLab\dragNdrop.c, 16) != 0)

{

        type "Failed to load dragNdrop.c!";

        return 0;

}
```

4.2.4 Identifying Errors

When you compile and link source files in Code Builder, the compiling and linking results are displayed in the Code Builder **Output** window.
If the compiling and linking was successful, the **Output** window lists the source files that were compiled. The **Done**! line indicates success.
If errors were encountered during the compiling and linking process, the **Output** window lists the file name, line number, and the error encountered. You can double-click on the error line in the **Output** window to activate the source file and position the cursor on the line of code containing the error.


Double-click here to activate this line (missing semicolon).

```
String getStrTest()

void mytest()
{
    printf("%s\n",getStrTest());
}
String getStrTest()
{
    String strB;
    double ff = PI;
    strB.Format("The value of PI is %f",ff);
    return strB;
}
```

```
compiling...
stringtest.c
C:\Program Files\OriginLab\Origin70\OriginC\stringtest.c[25] :Error, sy
C:\Program Files\OriginLab\Origin70\OriginC\stringtest.c[25] :Error, sy
```

4.3 Debugging

4.3.1 Debugging in Code Builder

Code Builder has features that allow you to debug your Origin C and LabTalk code. You can set and remove breakpoints, step through your code one statement at a time, step into and out of functions, and monitor the values of variables. Debugging is turned on by default. You can turn debugging on or off using the Enable Breakpoints item on the Debug menu. If there is a check mark next to the item then debugging is turned on.
More details of the debugger are covered in the Code Builder User Guide, which can be found in Origin under *Help: Programming: Code Builder*.

4.3.2 Macros for Debugging

Origin C allows users to define multi-parameter macros which have many uses. Two such uses, which help while debugging, are the ASSERT macro and macros containing debug statements. The following sections contain example code and discuss these macros in more detail. The example file DebugMacros.c can be found in the *\Samples\Origin C Examples\Programming Guide\Introduction to Origin C* subfolder of Origin.

The ASSERT Macro

Origin C provides an ASSERT macro to identify logic errors during program development. The Origin C ASSERT macro has the following syntax:

```
ASSERT(condition)
```

When a function is called that includes an ASSERT, the ASSERT launches a message box when the **condition** evaluates to **false** (zero). The diagnostic message indicates the source file and line number where the assertion failed. Thus, if your function expects a variable value to be non-zero, you could add the following line of code to alert you if the variable value is zero.

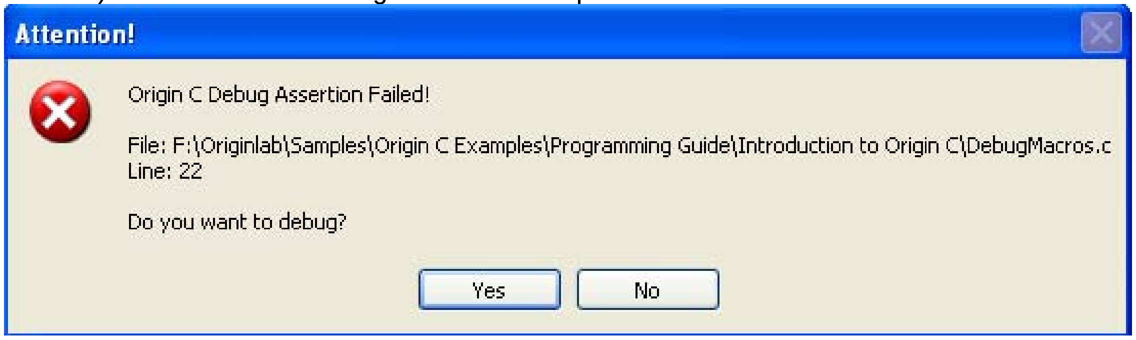
```
ASSERT(myVar != 0);
```

In the following example, if the size of the vector **bb** is not equal to the size of the vector **aa**, then a diagnostic message box will launch to indicate assert failure.

```
void AssertMacro(int imax = 10)
{
       vector<int> aa(imax);
       vector<int> bb;

       for (int ii = 0; ii < aa.GetSize(); ii++)
            aa[ii] = ii;
       bb = 10 * aa;

       // imax += 1;
       ASSERT(bb.GetSize() == imax);
```

```
        for (ii = 0; ii < bb.GetSize(); ii++)
             printf("bb[%d]=%d \n", ii, bb[ii]);
}
```

If you run the above function with the line **imax+=1**; commented out then no ASSERT message box appears. However, if you uncomment the line **imax+=1**; (and rebuild) the ASSERT message below is outputted.

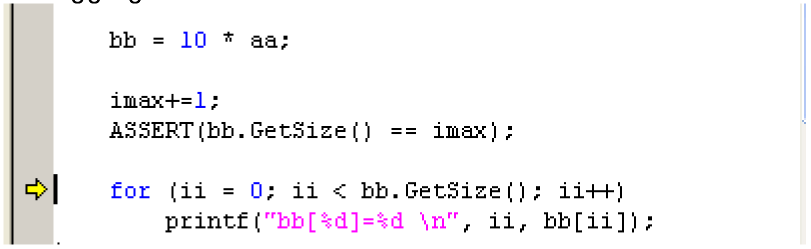


Origin C Debug Assertion Failed!

File: F:\Originlab\Samples\Origin C Examples\Programming Guide\Introduction to Origin C\DebugMacros.c
Line: 22

Do you want to debug?

Clicking Yes will open the appropriate source file (DebugMacros.c) in the Code Builder text editor and stop at the line generating the ASSERT message box for debugging.



```
    bb = 10 * aa;

    imax+=1;
    ASSERT(bb.GetSize() == imax);

    for (ii = 0; ii < bb.GetSize(); ii++)
        printf("bb[%d]=%d \n", ii, bb[ii]);
```

Debug Statements

Many programmers use output statements while developing code to indicate program flow and display the values of variables at key moments.

Create an Output Macro

A convenient debugging technique is to define an output macro and then place that macro throughout your code as shown below.

```
#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.");
}
```

Comment the Debug Macro Body

During the development cycle the body of the macro can remain defined as above causing the desired debug messages to be shown on the message box. However, once development is complete (or at least stable) the macro can be redefined as below causing the debug statements to disappear.

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

Commenting out the body of the **DBG_OUT** macro (and rebuilding) causes the debug statements to disappear without having to remove the many possible instances of its use, saving them for possible reuse in the future. Should the code ever need to be modified or debugged again the body of the macro can simply be uncommented.

4.4 Using Compiled Functions

Once Origin C functions have been compiled, linked and loaded, they are ready to be used in Origin. This means calling the function by its name and providing the necessary arguments from any location in Origin that accepts LabTalk script commands. Common locations include the script window, the command window, or a custom button in the Origin GUI. Running Scripts chapter of the LabTalk Scripting Guide details all of the locations in Origin from which script, and therefore Origin C functions, can be used.

4.4.1 Accessing Origin C Functions from LabTalk Script

Origin C functions can be called from other Origin C functions and from LabTalk scripts. This section talks about how to control the access to Origin C functions from LabTalk.
For information about accessing LabTalk from your Origin C code, refer to the Accessing LabTalk chapter.

Origin C Access from LabTalk

You can control LabTalk access to your Origin C code by putting a pragma statement in your Origin C code before your function definitions.

```
#pragma labtalk(0) // Disable OC functions in LabTalk
void foo0()
{
}

#pragma labtalk(1) // Enable OC functions in LabTalk (default)
void foo1()
{
}

#pragma labtalk(2) // Require '''run -oc''' LabTalk command
void foo2()
{
}
```

The above code prevents foo0 from being called from LabTalk, allows foo1 to be called from LabTalk, and allows foo2 to be called from LabTalk using the **run -oc** command. If you were to comment out the second pragma, then both foo0 and foo1 would be prevented from being called from LabTalk. This is because a single pragma statement applies to all functions after the pragma and up to the next pragma or the end of the file.
There is also a LabTalk system variable that controls LabTalk access to all Origin C functions. The variable is **@OC**, and it defaults to 1, which enables access. Setting the variable to 0 disables access.

Listing Functions that can be Called from LabTalk

The LabTalk **list** command can be used to output all the names of Origin C functions that can be called from LabTalk. Options let you modify which type of functions is listed:

```
list f;  // List functions callable from LabTalk
list fs; // List only those returning a string
list fv; // List only those returning a vector
list fn; // List only those returning a numeric
list fo; // List only those returning void
```

Note that setting **@OC=0** will make Origin C functions effectively invisible to LabTalk, such that the **list f** command will give no result.

Passing Arguments to Functions

LabTalk script does not support all of the data types used internally by Origin C. The following table lists the LabTalk variable types that should be passed (or returned) when calling an Origin C Function with the given argument (or return) type. The final column indicates whether or not that argument type can be passed by reference.

Origin C	LabTalk	Pass By Reference?
int	int	Yes
double	double	Yes
string	string	Yes
bool	int	No
matrix	matrix range	Yes
vector<int>	dataset	Yes
vector<double>	dataset	Yes
vector<complex>	dataset	No
vector<string>	dataset, string array*	No

* string arrays cannot be passed by reference

As the table above indicates, arguments of Origin C functions of type **string**, **int**, and **double** may be passed by value or by reference from LabTalk. Note, however, that the Origin C function must be written for the type of pass being performed.

Passing by Value

Below are examples of passing arguments by value from LabTalk to Origin C. The format for each example is to give the Origin C function declaration line, and then the LabTalk code used to call it. The Origin C function body is left out since it is unimportant for demonstrating variable passing.
The simple case of a function accepts an argument of type double and returns a double.

```
double square(double a)  // Origin C function declaration
double dd = 3.2;         // LabTalk function call
double ss = square(dd);
ss =;                    // ss = 10.24
```

Here, an Origin C function that takes a vector argument and returns a vector, is called by LabTalk using data set variables, or ranges that are assigned to data types.

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

Can be called three ways from LabTalk:

```
dataset dA, dB;
dB = Col(B);
dA=PassStrArray(dB);

Col(A)=PassStrArray(Col(B));  // Or, use Col directly, Col = dataset

// Or, LabTalk ranges may also be used
range ra = [Book1]1!1, rb = [Book1]1!2;
ra = PassStrArray(rb);
```

Passing by Reference

For the Origin C function below, note the ampersand **&** character in the argument declaration, indicating that the argument will be passed by reference.

```
double increment(double& a, double dStep)
double d = 4;
increment(d, 6);
type -a "d = $(d)";  // d = 10
```

The following example demonstrates some arguments being passed by reference and others being passed by value.

```
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);

//Or use a data set from a column; be sure to put data in Col(A)
get_min_max_double_arr(Col(A), dMin,  dMax);
```

The following example shows passing a LabTalk matrix range variable by reference to an Origin C function.

```
// set data from vector to matrix
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);
```

Precedence Rules for Functions with the Same Name

When a user-defined or global Origin C function has the same name as a built-in LabTalk function, the Origin C function has higher precedence, except when using LabTalk vector notation.

Precedence:

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

Thus, LabTalk functions like Normal and Data (which return a range of values and are thus used in vector notation) would have higher precedence than Origin C functions of the same name. In all other cases, the Origin C function is called.

4.4.2 Defining Functions for the Set Values Dialog

You may want to define a function using Origin C, that will appear in the **Set Values** menu of either a column or a matrix.

If an Origin C function is built as part of an Origin project---either automatically by being placed in the Project or System folder of Code Builder, or manually by building a function in the User folder---it will be available in the **User-Defined** section of the **F(x)** menu in the **Set Values** dialogs (for both Columns and Matrices). To assign a function to a different section of the **F(x)** menu, issue a pragma containing the new section name as part of the function header. For instance, the following code will add function **add2num** to the Math section and function **mean2num** to the Statistics section:

```
#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;
}
```

In this way, many functions can be defined in a single source file and, upon building, be immediately available in the desired locations of the **F(x)** menu.

Functions to be added to the **F(x)** menu must conform to the following additional restrictions:

- The return type of the function cannot be void
- The function should not have reference or pointer (&) for argument type

4.5 Distributing Origin C Code

4.5.1 Distributing Source Code

Origin users can share Origin C source code with one another by distributing either the source files themselves (.C or .CPP) or preprocessed files (.OP).
If it is not necessary for others to see your application's source code, it is highly recommended that you distribute preprocessed files (.OP) instead of the source files (.C or .CPP). See the Preprocessed Files (*.op) in the **Create and Edit an Origin C File** section for more information.

4.5.2 Distributing Applications

After creating an application, you can distribute it as a single package file to other Origin users.
Use Package Manager to package all the application files into a single package file (.OPX). Note that when adding your application files into the package, be sure to add the preprocessed files (.OP) or the source files (.C or .CPP). It is not necessary to add both.
Users can install your application by dropping the package file directly into Origin.
The following is an example that shows how to package all the application files into one OPX file. The user can drop the package file into Origin to install, then click a button to run the source file.

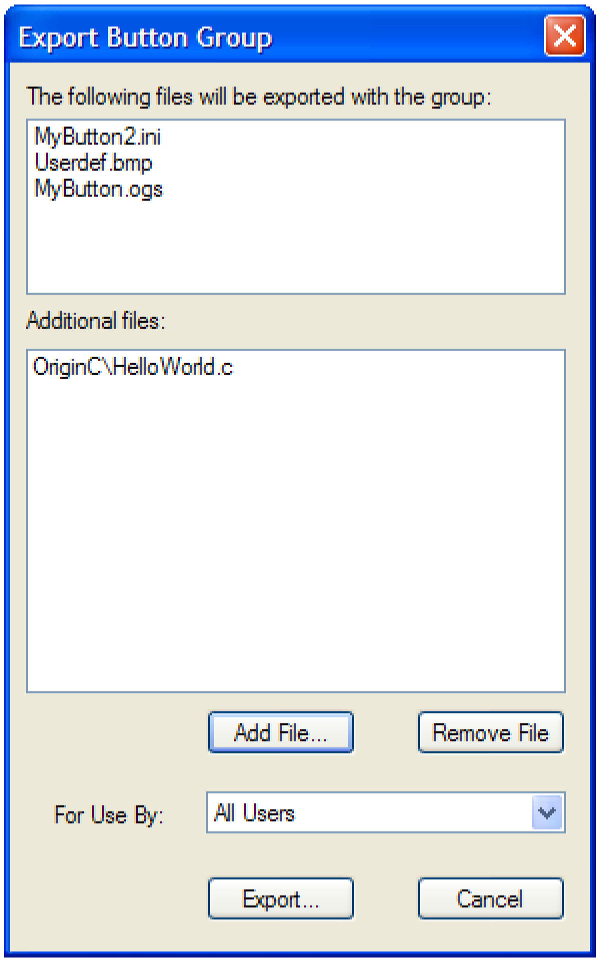
1. Prepare an Origin C source file. In Code Builder, choose menu File->New to create a new c file named MyButton.c, copy the following code to it and save it to the User File Folder\OriginC subfolder.

```
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. Create an OGS file named MyButton.ogs to load the Origin C source file and call function. Copy the following and save it to the User File Folder.

```
[Main]

        if(0 == Run.LoadOC(%Y\OriginC\MyButton.c))

        {

                OnButtonClick;

        }
```

3. In the Origin menu, choose View->Toolbars. In the Customize Toolbar dialog, choose the Button Groups tab, and click New to open the Create Button Group dialog. Set *MyButton* as the Group Name, keep Number of Buttons as 1, choose the Userdef.bmp file from the User File Folder as Bitmap, and click the OK button. In the ensuing Save As dialog, click the Save button to save the MyButton.ini file to the default path.
4. In the Customize Tool dialog, click to choose the ▦ button, then click Settings to open a Button Settings dialog. Choose MyButton.ogs as the File Name, type Main in for Section Name, and click OK to close the dialog.
5. Click Export to open the Export Button Group dialog, then click Add File and choose the above MyButton.c file.



MyButton2.ini
Userdef.bmp
MyButton.ogs

OriginC\HelloWorld.c

For Use By: All Users

6. Click Export, then in the Save As dialog click Save to save the MyButton.OPX file to the specified folder.
7. Choose menu Tools->Package Manager, and in the dialog that opens, choose File->Open to open the MyButton.OPX file. Put the script *Run.LoadOC(%Y\OriginC\HelloWorld.c);* into *LabTalk Script->After Installation* in gird view to load the Origin C source file. This script will be run when you drop OPX into Origin to install this application.

5 Workbooks and Matrixbooks

The Origin C **WorksheetPage** and **MatrixPage** classes are for working with Origin workbooks and matrix books respectively. Each book contains a collection of sheets and each sheet contains a collection of **Column**s or **MatrixObject**s.

Creating a Workbook / Matrixbook

The **Create** method is used for creating new books.

```
// create a hidden workbook using the ColStat template
WorksheetPage wksPg;
wksPg.Create("ColStat", CREATE_HIDDEN);

MatrixPage matPg;
matPg.Create("Origin"); // create a matrix book using the Origin
template
```

Accessing a Workbook / Matrixbook

There are multiple ways to access an existing workbook or matrix book. The methods used are the same for both workbooks and matrix books.
The **Project** class contains a collection of all the workbooks or matrix books in the project. The following example shows how to loop through them.

```
foreach(WorksheetPage wksPg in Project.WorksheetPages)
    out_str(wksPg.GetName()); // output workbook name

foreach(MatrixPage matPg in Project.MatrixPages)
    out_str(matPg.GetName()); // output matrix book name
```

You can access a workbook or matrix book by passing its index to the **Item** method of the **Collection** class.

```
MatrixPage matPg;
matPg = Project.MatrixPages.Item(2);
if( matPg ) // if there is a 3rd matrix book
    out_str(matPg.GetName()); // output matrix book name
```

You can access a workbook or matrix book by passing its name to the class constructor.

```
WorksheetPage wksPg("Book1");
if( wksPg ) // if there is a workbook named "Book1"
    wksPg.SetName("MyBook1"); // rename the workbook
```

Deleting a Workbook / Matrix Book

All of Origin C's internal classes are derived from the **OriginObject** class. This class has a **Destroy** method that is used to destroy the object. Calling this method on a workbook or matrix book will destroy it, all the sheets in the workbook or matrix book, and all the columns or matrix objects in each sheet.

```
WorksheetPage wksPg;
```

```
wksPg = Project.WorksheetPages.Item(0); // get first workbook in
project
if( wksPg ) // if there is a workbook
    wksPg.Destroy(); // delete the workbook
```

Clone a Workbook/Matrix Book

The **WorksheetPage** class (for a Workbook), **MatrixPage** class (for a Matrixbook) and **GraphPage** class (for a Graph window) are all derived from the **Page** class. This class has a **Clone** method that is used to clone the source page.

```
// Duplicate "Book1", "MBook1" and "Graph1" window with data and style
// Before calling make sure these windows exist
WorksheetPage wksPage("Book1");
WorksheetPage wksPage2 = wksPage.Clone();

MatrixPage matPage("MBook1");
MatrixPage matPage2 = matPage.Clone();

GraphPage gp("Graph1");
GraphPage gp2 = gp.Clone();
```

5.1 Worksheets

Origin C provides the **Worksheet** class for working with the worksheets in a **WorksheetPage**. While a workbook contains a collection of worksheets, a worksheet contains a collection of **Column**s. The **Worksheet** class is derived from the **Layer** class.

5.1.1 Adding and Deleting Worksheets

Add a worksheet to a workbook using the **AddLayer** method of the **WorksheetPage** class.

```
// Access the workbook named "Book1"
WorksheetPage wksPage("Book1");

// Add a new sheet to the workbook
int index = wksPage.AddLayer("New Sheet");

// Access the new worksheet
Worksheet wksNew = wksPage.Layers(index);

// Set the new worksheet to be active
set_active_layer(wksNew);
```

Use the **Destroy** method to delete a worksheet.

```
Worksheet wks = Project.ActiveLayer();
if( wks ) // If the active layer is a worksheet
    wks.Destroy(); // Delete the worksheet
```

5.1.2 Accessing Worksheets in a Workbook

There are two ways to access a worksheet by its name. You can pass the layer's full name to the constructor or to the Attach method. The layer's full name contains the page name in square brackets followed by the layer name.

```
// Assume wksPage is a valid WorksheetPage holding the sheet we want to
access.
string strFullName = "[" + wksPage.GetName() + "]" + "Sheet1";

// If book and sheet name are known, the string can be constructed
manually.
string strFullName = "[Book5]Sheet4";
```

With the full layer name we can now access the worksheet.

```
// Construct a new Worksheet instance and attach it to the named sheet.
Worksheet wks1(strFullName);

// Attach an existing Worksheet instance to the named sheet.
wks2.Attach(strFullName);
```

A workbook contains a collection of worksheets. You can loop through all the worksheets in a specified workbook using the **foreach** statement.

```
WorksheetPage wksPage("Book1");
foreach(Layer wks in wksPage.Layers)
    out_str(wks.GetName());
```

You can also access a specified worksheet by its name or index.

```
//assume there are at least two worksheets on the page Book1,
//and they are named Sheet1 and Sheet2 separately.
WorksheetPage wksPage("Book1");
Worksheet wksFirst = wksPage.Layers(0); //by index
Worksheet wksSecond = wksPage.Layers("Sheet2"); //by name
```

5.1.3 Reorder Worksheets

The **Reorder** method allows you to change the position of a worksheet in a workbook.

```
// This example assumes the active workbook contains two sheets

// Get the active page from the active layer
WorksheetPage wksPage;
Worksheet wks = Project.ActiveLayer();
if( wks )
    wksPage = wks.GetPage();

// Move the 2nd worksheet to the 1st position
if( wksPage.Reorder(1, 0) )
    out_str("Reorder sheets successfully");
```

5.1.4 Copying Worksheet

The **Page::AddLayer** method is used to copy a layer from one page to another, and can be used with GraphPage, WorksheetPage or MatrixPage.

The following example shows how to drag all worksheets from the active folder to merge into the active workbook.

```
WorksheetPage wksPageDest = Project.Pages();
if( !wksPageDest ) // no active window or active window is not a
worksheet
        return;

bool bKeepSourceLayer = false; // delete source layer after copying
Folder fld = Project.ActiveFolder();
foreach(PageBase pb in fld.Pages)
{
    WorksheetPage wbSource(pb);
    if(!wbSource)
        continue;//not a workbook

    if(wbSource.GetName() == wksPageDest.GetName())
        continue;//skip our destination book

    // copy worksheet to destination book and delete it from source
book
    foreach(Layer lay in wbSource.Layers)
    {
        Worksheet wks = lay;
        wksPageDest.AddLayer(wks, 0, bKeepSourceLayer);
    }
    wbSource.Destroy();// destroy the empty workbook
}
```

5.1.5 Formatting a Worksheet

A worksheet can be formatted programmatically using a theme tree. The example below demonstrates obtaining and saving an existing theme tree:

```
// get format tree from worksheet
Worksheet wks = Project.ActiveLayer();

Tree tr;
tr = wks.GetFormat(FPB_ALL, FOB_ALL, TRUE, TRUE);
out_tree(tr); // Output tree to Script window
```

Or, you may construct a theme tree as in the following three steps. First, create a worksheet and insert some data:

```
// Create worksheet
Worksheet wks;
wks.Create("Origin");
wks.SetCell(0, 0, "abc"); // Put text to (0, 0) cell

// Establish data range to apply formatting:
DataRange dr;
int r1 = 0, c1 = 0, r2 = 4, c2 = 1;
dr.Add("Range1", wks, r1, c1, r2, c2);
```

Second, construct the tree using the range information and provide values for desired properties:

```
Tree tr;
// Setup the range that the format want to apply
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;

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

// Alignment of text in cell, 2 for center
tr.Root.RangeStyles.RangeStyle1.Style.Alignment.Horizontal.nVal = 2;

// The font size of text
tr.Root.RangeStyles.RangeStyle1.Style.Font.Size.nVal = 11;

// The color of text
tr.Root.RangeStyles.RangeStyle1.Style.Color.nVal = SYSCOLOR_BLUE;
```

Third, apply the formatting to the data range:

```
// Apply the format to the specified data range
if( 0 == dr.UpdateThemeIDs(tr.Root) ) // Returns 0 for no error
{
        bool bRet = dr.ApplyFormat(tr, true, true);
}
```

5.1.6 Merge Cells

We can use Origin C code to merge Worksheet cells with the specified range. The selected range can be data area or column label area. If you want to merge label cells, just change *bLabels* to **true** in the following code.

```
Worksheet wks;
wks.Create("Origin");

//Define a Grid and attach it to the worksheet
Grid gg;
gg.Attach(wks);

// to merge the first two rows in two columns
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());
```

5.2 Columns

Origin C provides the Column class for handling the columns in a worksheet. A Column object is usually used to control the style, format and data type of the dataset, which is contained in the column. Example codes, demonstrating how to use the Column class, are provided in the following sections.

5.2.1 Dimensions

Add Column to Worksheet

```
// Add column with default name
int nColIndex = wks.AddCol();
// Add column with name
string strName;
int nColIndex = wks.AddCol("AA", strName); // Returns the index of
column

// If the column named AA already exist, name enumeration automatically
out_str(strName);

Column col(wks, nColIndex); // Construct column object by column index
```

Delete Column from Worksheet

```
// Delete the column by index
wks.DeleteCol(0);
```

Insert Column into Worksheet

```
// Insert a new column as the first column
int nPos = 0; // The position to insert
string strNewCreated; // the real name of the new column

// The name will be auto enumerated if name MyCol already existed
if( wks.InsertCol(nPos, "MyCol", strNewCreated) )
{
        printf("Insert column successfully, name is %s\n",
strNewCreated);
}
```

Set Number of Rows and Columns

```
// Set the number of rows and columns, and data will be kept.
// If want to add a lots of columns and rows at once time, better use
SetSize
int nNumRows = 100;
int nNumCols = 20;
wks.SetSize(nNumRows, nNumCols);

// If want to change the number of rows but keep the number of columns,
// can use -1 replace. For example:
```

```
wks.SetSize(nNumRows, -1);

// The same usage also used to change column number and keep row
number.
```

5.2.2 Data Type, Format, SubFormat

Get & Set Data Type

```
Worksheet wks = Project.ActiveLayer();
Column col(wks, 0);

// Get column type, can be:
// 0: Y
// 1: None
// 2: Y Error
// 3: X
// 4: L
// 5: Z
// 6: X Error
int nType = col.GetType();
out_int("Type: ", nType);
// Set column type. See more define OKDATAOBJ_DESIGNATION_* in
oc_const.h
col.SetType(OKDATAOBJ_DESIGNATION_Z);
```

Get & Set Data Format

```
// Get and set data format
// The default format of column is OKCOLTYPE_TEXT_NUMERIC.
// Set the format of column to Date
if( OKCOLTYPE_DATE != col.GetFormat() )
{
	col.SetFormat(OKCOLTYPE_DATE);
}
```

Get & Set Data Subformat

```
// Get and set data subformat
// The options of the sub format will be different according to the
above format,
// numeric, date, time and so on.
if( LDF_YYMMDD != col.GetSubFormat() )
{
	col.SetSubFormat(LDF_YYMMDD);
}
```

5.2.3 Data Manipulation

Get & Set Numeric Data Values from Column

```
// Attach to the first column, make sure the format of the column is
// Text & Numeric(default) or Numeric.
Column col(wks, 0);

// Here assume the data type of the column is double.
// Other numeric data type supported, for example, int, short, complex.
vector<double>& vec = col.GetDataObject();

// Append 100 at the end of this column
vec.Add(100);
```

Or we can use a Dataset object to get and set numeric data for a column. For example:

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

Dataset ds(wks, 1);

for(int ii=0; ii<ds.GetSize(); ii++)
    out_double("", ds[ii]);
```

Get & Set String Values from Column

```
Column col(wks, 0); // Attach to the first column

// Get string array from column
vector<string> vs;
col.GetStringArray(vs);

// Put string array back to column
vs.Add("test");
col.PutStringArray(vs);
```

5.2.4 Column Label

Worksheet column labels support Long Name, Units, Comments, Parameters and User-Defined labels. We can use Origin C code to show/hide labels or to add text to the specified column label.

```
Worksheet wks;
wks.Create();

Grid gg;
gg.Attach(wks);

// if Parameters lable not show, show it.
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");

// put text to Parameters label for two columns.
wks.Columns(0).SetExtendedLabel("Param A", RCLT_PARAM);
wks.Columns(1).SetExtendedLabel("Param B", RCLT_PARAM);
```

RCLT_PARAM is the type of Parameters column label, other types see *OriginC\system\oc_const.h* file ROWCOLLABELTYPE enum.

5.2.5 Setting Value by Formula

The **DataObject::SetFormula** and **DataObject::ExecuteFormula** methods are used to set column/matrix values, which is the same as setting values in the **Set Values** dialog. The following example is of creating a worksheet with three columns, and then setting values by a formula to each column.

```
Worksheet wks;
wks.Create("origin", CREATE_VISIBLE);
wks.AddCol();

// set value to the first column
Column colA;
colA.Attach(wks, 0);
colA.SetFormula("5*(i-1)");
colA.ExecuteFormula();

// for the next two columns we will set Recalculate = Auto
Column colB;
colB.Attach(wks, 1);
colB.SetFormula("sin(4*col(A)*pi/180)", AU_AUTO);
colB.ExecuteFormula();

// using declared variables in Before Formula Script
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();
```

5.3 Matrixsheets

Origin C provides the MatrixLayer class for working with a matrix sheet. A matrix sheet contains a collection of matrix objects.

5.3.1 Adding and Deleting a Matrix Object

```
// add matrix object to sheet
MatrixLayer ml = Project.ActiveLayer(); // Get active matrix sheet

int nNum = 1; // the number of added matrix objects
int nPos = -1; // -1, add as the end
int nDataType = -1; // Optional, -1 as default for double type.
int index = ml.Insert(nNum, nPos, nDataType); // Returns the index of
the first one
// delete matrix object from sheet
MatrixLayer ml = Project.ActiveLayer(); // Get active matrix sheet

// Delete two matrix objects from the beginning
int nPos = 0;
int nNum = 2;
ml.Delete(nPos,nNum);
```

5.3.2 Accessing a Matrix Object

```
// Attach to one matrix page by name
MatrixPage matPage("MBook3");

// Attach to the sheet named MSheet1 from matrix page
// Also support get sheet from matrix page by index
MatrixLayer ml1 = matPage.Layers("MSheet1");

// Get a matrix object from sheet by index
MatrixObject mo = ml1.MatrixObjects(0);

// The data type of matrix object must keep consistent with the matrix
window
if( FSI_SHORT == mo.GetInternalDataType() )
{
        matrix<short>& mat = mo.GetDataObject();
}
```

5.3.3 Setting View as Data or Image

```
// set image view
MatrixLayer ml = Project.ActiveLayer(); // Get active matrix sheet

int nImgIndex = 0;
MatrixObject mo = ml.MatrixObjects(nImgIndex);

if( !mo.IsImageView() )
{
        BOOL bAllObjs = FALSE;
        ml.SetViewImage(TRUE, bAllObjs, nImgIndex);
}
```

5.3.4 Formatting a Matrix Sheet

A matrix sheet can be formatted programmatically using a theme tree.
The example below formats a block of cells in the active matrix sheet to have a blue background and light-magenta text.

```
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); // first row, col, last row, col
if( 0 == dr.UpdateThemeIDs(tr.Root) )
    dr.ApplyFormat(tr, TRUE, TRUE);
```

5.3.5 Matrix Cell Text Color

The next example shows how to get and set the text color of a cell.

```
// Wrap the 'set' code into a simpler utility function.
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);
}

// Wrap the 'get' code into a simpler utility function.
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;
}

// Simple function for testing the above utility functions.
void testCellTextColor(int nRow = 3, int nCol = 4)
{
    MatrixLayer ml = Project.ActiveLayer();
        // nRow, nCol use LT/GUI indexing, 1-offset, but OC is 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);
}
```

5.4 Matrices

The Origin C MatrixObject class allows you to handle a matrix, including dimension, data type, data format, setting values, etc. Please refer to the related sections below.

5.4.1 Dimensions

Get Number of Rows and Columns in Matrix

```
// get num rows and cols
MatrixLayer ml = Project.ActiveLayer(); // Get active matrix sheet
MatrixObject mo = ml.MatrixObjects(0); // Get the first matrix object

int nNumRows = mo.GetNumRows();  // Get the row number
int nNumCols = mo.GetNumCols();  // Get the column number
```

Set Dimensions of Matrix

```
// set num rows and cols
MatrixLayer ml = Project.ActiveLayer(); // Get active matrix sheet
MatrixObject mo = ml.MatrixObjects(0); // Get the first object

int nNumRows = 5, nNumCols = 5;
mo.SetSize(nNumRows, nNumCols);  // Set dimensions by 5x5
```

5.4.2 Data Type and Format

Get & Set Data Type

```
// get and set data type
MatrixLayer ml = Project.ActiveLayer(); // Get active matrix sheet
MatrixObject mo = ml.MatrixObjects(0);

if( mo.GetInternalDataType() != FSI_BYTE ) // Get data type
{
        // OCD_RESTORE to backup the data and
        // attempt to restore it after changing type
        DWORD dwFlags = OCD_RESTORE;
        mo.SetInternalDataType(FSI_BYTE, dwFlags); // Set data type
}
```

Get & Set Data Format

```
// get and set data format
MatrixLayer ml = Project.ActiveLayer(); // Get active matrix sheet
MatrixObject mo = ml.MatrixObjects(0);

int nFormat = mo.GetFormat(); // Only OKCOLTYPE_NUMERIC( = 0) supported
mo.SetFormat(OKCOLTYPE_NUMERIC);
```

5.4.3 Labels

A matrix label includes a Long Name, Units, and Comments for X, Y, Z. The labels of X and Y are for all matrix objects in the matrix sheet, the label of Z is for each matrix object. The following code shows how to get and set the labels.

Set XY Labels

```
MatrixPage mp("MBook1");
MatrixLayer ml = mp.Layers(0); // the first matrix sheet

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";

// Note, set format on matrix sheet for XY labels.
if( 0 == ml.UpdateThemeIDs(tr.Root) )
        ml.ApplyFormat(tr, true, true);
```

Get XY Labels

```
MatrixPage mp("MBook1");
MatrixLayer ml = mp.Layers(0); // the first matrix sheet

// Note, get XY labels from matrix sheet, not matrix object.
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);
```

Set Z Labels

```
MatrixPage mp("MBook1");
MatrixLayer ml = mp.Layers(0); // the first matrix sheet
MatrixObject mo = ml.MatrixObjects(0);// the first matrix object

// construct format tree and assign string value to tree nodes
Tree tr;
tr.Root.LongName.strVal = "Z Long Name";
tr.Root.Unit.strVal = "Z Units";
tr.Root.Comment.strVal = "Z Comment";

// Note, here apply format on matrix object to set Z labels, not matrix
sheet.
if( 0 == mo.UpdateThemeIDs(tr.Root) ) // add id for each tree node
        mo.ApplyFormat(tr, true, true);        // do apply
```

Get Z Labels

```
MatrixPage mp("MBook1");
MatrixLayer ml = mp.Layers(0); // the first matrix sheet
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() )// if not empty
        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);
```

5.4.4 Data Values

Multiply Matrix by Constant

```
MatrixLayer ml = Project.ActiveLayer(); // Get active matrix sheet
MatrixObject mo = ml.MatrixObjects(0); // Get the first matrix object

//Get the reference of the internal data object of matrix window.
//Here assume data type of the matrix is double.
matrix<double>& mat = mo.GetDataObject();

// multiply 10 for each data in matrix, this change also effect on
window
mat = mat * 10;
```

Dot Multiply Two Matrix

```
// Attach to two matrix pages
MatrixPage matPage1("MBook1");
MatrixPage matPage2("MBook2");
if( !matPage1 || !matPage2 )
        return;

// Get the matrix sheet from page by name or index
MatrixLayer matLayer1 = matPage1.Layers("MSheet1");
MatrixLayer matLayer2 = matPage2.Layers(1); // get the second sheet
if( !matLayer1 || !matLayer2 )
        return;

// Get matrix object from matrix sheet by index, name is not allowed.
MatrixObject mo1 = matLayer1.MatrixObjects(0);
MatrixObject mo2 = matLayer2.MatrixObjects(0);

// Get the reference of the internal data object of matrix window
matrix<double>& mat1 = mo1.GetDataObject();
matrix<double>& mat2 = mo2.GetDataObject();

// Prepare new matrix window
MatrixPage matPageNew;
matPageNew.Create("Origin");
MatrixLayer mlNew = matPageNew.Layers(0);
```

```
MatrixObject moNew = mlNew.MatrixObjects(0);
matrix<double>& matNew = moNew.GetDataObject();

// Copy values from mat1 to new matrix
matNew = mat1;

// Multiply two matrices element by element and put result
// to a newly created matrix window
matNew.DotMultiply(mat2);
```

5.4.5 Setting Value by Formula

The **DataObject::SetFormula** and **DataObject::ExecuteFormula** methods are used to set column/matrix values, which is the same as setting values in the **Set Values** dialog. The example below shows how to set values to a matrix object by formula.

```
// new a matrix window
MatrixPage matPage;
matPage.Create("Origin");
MatrixLayer ml = matPage.Layers(); // get active matrix sheet

// set formula and execute
MatrixObject mo = ml.MatrixObjects(0); //get first matrixobject
mo.SetFormula("sin(i) + cos(j)");
mo.ExecuteFormula();
```

5.5 Virtual Matrix

You can construct a virtual matrix from a worksheet window. Pick separate data ranges from the worksheet for X, Y, Z data of the virtual matrix. If you do not specify X and Y data, it will automatically use default data. The following code shows how to construct a virtual matrix from an active worksheet window, and then plot this virtual matrix on a graph.

```
// before running, make sure there is active worksheet window with
data.
// For example, new a worksheet window, import XYZ Random Gaussian.dat
from
// Origin folder Samples\Matrix Conversion and Gridding subfolder to
worksheet.
Worksheet wks = Project.ActiveLayer();

int r1, r2;
int c1 = 0, c2 = 2;
wks.GetBounds(r1, c1, r2, c2);

// construct a data range object only with Z data, X and Y data will be
auto
// assigned.
DataRange dr;
dr.Add("Z", wks, r1, c1, r2, c2);
```

```
MatrixObject mo;
mo.Attach(dr);

int nRows = mo.GetNumRows();
int nCols = mo.GetNumCols();

// get the default x, y range
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();
```

If you want to assign X and Y data then the data should be monotone. The following example shows how to construct a virtual matrix with an XYZ data range.

```
// Assume the active layer is a worksheet with 5 columns of data.
Worksheet wks = Project.ActiveLayer();

// Get min and max row indices for columns 0 to 4.
int r1, r2, c1 = 0, c2 = 4;
wks.GetBounds(r1, c1, r2, c2);

// Create a data range object with XYZ data.
DataRange dr;
dr.Add("X", wks, 0, 1, 0, c2); // First row except the first cell
dr.Add("Y", wks, 1, 0, r2, 0); // First column except the first cell
dr.Add("Z", wks, 1, 1, r2, c2);

MatrixObject mo;
mo.Attach(dr);
```

5.6 Data Manipulation

5.6.1 Getting Worksheet Selection

Worksheet::GetSelectedRange can be used to get one or multiple selected data ranges from a worksheet. The following code shows how to get data from one column by worksheet selection. This function returns range type, like one column, one row, whole worksheet, etc.

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

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

```
if( WKS_SEL_ONE_COL & nRet ) // exactly one column selected
{
        // construct a data range object by selection
        DataRange dr;
        dr.Add("X", wks, r1, c1, r2, c2);

        // get data from the selected column
        vector vData;
        dr.GetData(&vData, 0);
}
```

5.6.2 Setting Display Range in Worksheet

If you want to set a display range in a Worksheet, you can use **Worksheet::SetBounds**, and it is the same as using the Set As Begin/End menu. The following code shows how to set a beginning and end for all columns in the current worksheet window.

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

// the beginning and end of rows
int begin = 9, end = 19;

// set beginning and end for all columns
int c1 = 0, c2 = -1; // -1 means end

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

5.6.3 Putting Large Dataset to Worksheet

In order to keep an Origin C function running efficiently when working with a large data set (e.g. 1000 columns) in a worksheet, use the steps below.

- Prepare the columns and rows before putting data into the worksheet.
- Use **Worksheet::SetSize**, don't use **Worksheet::AddCol** to set the size.
- Set the size on an empty worksheet, meaning no columns and rows, since otherwise Origin will need to check the short names of the existing columns to avoid duplicate names when adding new columns, and this could cost you lots of time. You can use **while(wks.DeleteCol(0));** to remove all columns to make an empty Worksheet.
- Put data into worksheet columns by buffer, **DataObject::GetInternalDataBuffer**.
- Keep Code Builder closed when running functions to improve the speed of execution.

See the following example codes:

```
// prepare worksheet size
Worksheet wks;
wks.Create("Origin");
while( wks.DeleteCol(0) );
int rows = 100, cols = 1000;
wks.SetSize(rows, cols);

// put data set into worksheet columns one by one
foreach(Column col in wks.Columns)
{
    col.SetFormat(OKCOLTYPE_NUMERIC);
    col.SetInternalData(FSI_SHORT);
    col.SetUpperBound(rows-1);//index of last row, 0 offset

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

    // OC loop is still slow, but you might pass this pointer to your
DLL
    // for much faster manipulation, here we just show that the pointer
works
    for(int ii = 0; ii < rows; ii++, psBuff++)
    {
        *psBuff = (ii+1) * (col.GetIndex()+1);
    }
    col.ReleaseBuffer(); // do NOT remember to call this
}
```

5.6.4 Accessing Embedded Graph in a Worksheet

Create a new graph and a new worksheet, and then embed the graph within one of the worksheet's cells:

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

Worksheet wks;
wks.Create();

int nOptions = EMBEDGRAPH_KEEP_ASPECT_RATIO | EMBEDGRAPH_HIDE_LEGENDS;

// Put the graph in worksheet cell (0, 0)
wks.EmbedGraph(0, 0, gp, nOptions);
```

Access a graph that is embedded within a worksheet; by name or by index:

```
// Get embedded graph from active worksheet
Worksheet wks = Project.ActiveLayer();

GraphPage gp;
gp = wks.EmbeddedPages(0); // Get embedded graph page by index

gp = wks.EmbeddedPages("Graph1"); // Get embedded graph page by name
```

5.6.5 Sorting Worksheet Data

Perform a row-wise sort of column data with the **Sort** method. For sorting a single column, use the **vectorbase::Sort** method:

```
// Sort column
// Before running, please keep active worksheet with two columns fill
with data.
// For example, import \Samples\Mathematics\Sine Curve.dat to
worksheet.
Worksheet wks = Project.ActiveLayer();
```

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

// After sort, the original relation for (x, y) will be broken.
vectorbase& vec = colY.GetDataObject();
vec.Sort();
```

To sort all columns in a worksheet, use the **Worksheet::Sort** method:

```
// Sort worksheet
// Before running, please keep active worksheet with two columns fill
with data.
// For example, import \Samples\Mathematics\Sine Curve.dat to
worksheet.
Worksheet wks = Project.ActiveLayer();

int nCol = 1; // Ascending sort all worksheet data on the second column
BOOL bIsAscending = true;
BOOL bMissingValuesSmall = TRUE; // Treat missing value as smallest
int r1 = 0, c1 = 0, r2 = -1, c2 = -1; // -1 means end for r2 and c2

// After sort, each (x, y) still keep the original relation
wks.Sort(nCol, bIsAscending, bMissingValuesSmall, r1, c1, r2, c2);
```

5.6.6 Masking Worksheet Data

The following code shows how to set a mask on the rows of data that are less than or equal to 0 for the specified column.

```
int nCol = 1;
Worksheet wks = Project.ActiveLayer();
Column col(wks, nCol);
vector vData = col.GetDataObject();

// to find all less than and equal 0 and return row index
vector<uint> vnRowIndex;
vData.Find(MATREPL_TEST_LESSTHAN | MATREPL_TEST_EQUAL, 0, vnRowIndex);

// construct a range including multiple subranges added by row and
column index
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);
}

// set mask on data range
dr.SetMask();
```

5.6.7 Extracting Data from Worksheet with LT Condition

Select worksheet data using the **Worksheet::SelectRows** method. Rows can be selected across many columns.

```
// Select data from a worksheet based on a condition;
// put the indices of the selected rows into a vector of type 'uint'.
Worksheet wks = Project.ActiveLayer();

// Check the worksheet data based on the condition expression and
// output the row index into 'vnRowIndices'.
// Define Labtalk range objects, 'a' = column 1, 'b' = column 2.
string strLTRunBeforeloop = "range a=1; range b=2";
string strCondition = "abs(a) >= 1 && abs(b) >= 1";
vector<uint> vnRowIndices; // This is output
int r1 = 0, r2 = -1; // The row range, -1 means the last row for r2

// Optional maximum number of rows to select, -1 indicates no limit
int nMax = -1;

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

There are two ways to highlight the selection. The first is to highlight the selected indices.

```
// Method 1 of show selection: highlight rows by vnRowIndices
Grid gg;
if( gg.Attach(wks) )
{
        // convert uint type vector to int type vector
    vector<int> vnRows;
    vnRows = vnRowIndices;

    gg.SetSelection(vnRows);
}
```

The second method of highlighting a data selection is to prescribe a fill color for the selected rows.

```
// Method 2 of show selection: fill color on the selected rows by
vnRowIndices
DataRange dr;

// Construct data ranges by the row indices in vnRowIndices.
for(int index=0; index<vnRowIndices.GetSize(); index++)
{
        // The following 0(1st col) and -1(last col) for all columns
        // "" for range name variable, not specified, default name will
be used
        dr.Add("", wks, vnRowIndices[index], 0, vnRowIndices[index], -
1);
}

Tree tr;
tr.Root.CommonStyle.Fill.FillColor.nVal = SYSCOLOR_BLUE; // fill color
= blue
tr.Root.CommonStyle.Color.nVal = SYSCOLOR_WHITE; // font color = white

if( 0 == dr.UpdateThemeIDs(tr.Root) ) // Return 0 for no error
{
        bool bRet = dr.ApplyFormat(tr, true, true);
}
```

5.6.8 Comparing Data in Two Worksheets

It may be useful to compare the number of rows or columns between two worksheets, or compare the data themselves. Get a row or column count from a worksheet with the **Datasheet::GetNumRows** and **Datasheet::GetNumCols** methods.

```
if( wks1.GetNumRows() != wks2.GetNumRows()
        || wks1.GetNumCols() != wks2.GetNumCols() )
{
        out_str("The two worksheets are not the same size");
        return;
}
```

Another way to perform a similar operation is to copy the data from each worksheet into a vector, and compare the size of the vectors.

```
// get all data from worksheet 1 columns one by one
vector vec1;
foreach(Column col in wks1.Columns)
{
        vector& vecCol = col.GetDataObject();
        vec1.Append(vecCol);
}

// get all data from worksheet 2 columns one by one
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;
}
```

To compare data elements themselves, use the **ocmath_compare_data** function on the vectors in the example above.

```
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");
}
```

5.7 Data Transformation

5.7.1 Worksheet Gridding

1. Run the following command in the Command Window to compile the nag_utils.c file and add it into the current workspace

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

2. Include header files in the Origin C file.

```
#include <wks2mat.h>

#include <Nag_utils.h>
```

3. Get XYZ data from the active worksheet XYZ columns.

```
// Construct XYZ data range from XYZ columns

XYZRange rng;

rng.Add(wks, 0, "X");

rng.Add(wks, 1, "Y");

rng.Add(wks, 2, "Z");



// Get XYZ data from data range objects to vectors

vector vX, vY, vZ;

rng.GetData(vZ, vY, vX);
```

4. Examine source data type, for example: regular, sparse.

```
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. Calculate the number of rows and columns for the result matrix window.

```
int nRows = 10, nCols = 10;

if( 0 == nMethod || 1 == nMethod ) // Regular or sparse

{

        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. Prepare the result matrix window.

```
// Prepare matrix window to put gridding result

MatrixPage mp;

mp.Create("origin"); // Create matrix window

MatrixLayer ml = mp.Layers(0); // Get the first matrix sheet

MatrixObject mo(ml, 0); // Get the first matrix object



mo.SetXY(xmin, ymin, xmax, ymax); // Set the from/to for X and Y

mo.SetSize(nRows, nCols); // Set the number of rows and columns
```

7. Do XYZ gridding with the different method types.

```
matrix& mat = mo.GetDataObject(); // Get data object from matrix
object
```

```
int iRet;

switch(nMethod)

{

case 0: // Regular

        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: // Sparse

        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: // Random(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: // Error.

        printf("--- Error: Other method type ---\n");

}
```

6 Graphs

The **GraphPage** class is for working with a graph window. There is a GraphPage object for each graph window. A GraphPage object contains a collection of layers. Each of these layers is a **GraphLayer** object.

Accessing an Existing Graph

There are multiple ways to access an existing graph. The methods used are the same as those used for workbooks and matrix books.

You can access a graph by passing its name to the class constructor.

```
GraphPage grPg("Graph1");
if( grPg ) // if there is a graph named "Graph1"
    grPg.SetName("MyGraph1"); // rename the graph
```

The **Project** class contains a collection of all the graphs in the project. The following example shows how to loop through the collection and output the name of each graph.

```
foreach(GraphPage grPg in Project.GraphPages)
    out_str(grPg.GetName()); // output graph name
```

You can access a graph by passing its zero-based index to the **Item** method of the **Collection** class.

```
GraphPage grPg;
grPg = Project.GraphPages.Item(2);
if( grPg ) // if there is a 3rd graph
    out_str(grPg.GetName()); // output graph name
```

Deleting a Graph

All Origin C's internal classes are derived from the **OriginObject** class. This class has a **Destroy** method that is used to destroy the object. Calling this method on a graph will destroy the graph, all the layers in the graph, and all the graph objects on each layer.

```
GraphPage grPg;
grPg = Project.GraphPages.Item(0); // get first graph in project
if( grPg ) // if there is a graph
    grPg.Destroy(); // delete the graph
```

6.1 Creating and Customizing Graph

6.1.1 Creating Graph Window

The **Create** method is used for creating new graphs.

```
GraphPage gp;
gp.Create("3D"); // create a graph using the 3D template
```

6.1.2 Getting Graph Page Format

```
GraphPage gp("Graph1");

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

6.1.3 Setting Graph Page Format

The following example code shows how to set page background color as a gradient in two colors.

```
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);
```

6.1.4 Getting Graph Layer Format

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

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

6.1.5 Setting Graph Layer Format

The following example code shows how to set the background of a graph layer object to Black Line format.

```
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);
```

6.1.6 Show Additional Lines

This example shows how to show additional lines, the Y=0/X=0 line, and the opposite line.

```
GraphLayer gl = Project.ActiveLayer();
Axis axesX = gl.XAxis;

axesX.Additional.ZeroLine.nVal = 1; // Show Y = 0 line
axesX.Additional.OppositeLine.nVal = 1; // Show X Axes opposite line
```

6.1.7 Show Grid Lines

This example shows how to set gridlines to show, and how to color them.
Color values can be an index into Origin's internal color palette or an RGB value. See Color in the Data Types and Variables section for more information about working with color values.

```
GraphLayer gl = Project.ActiveLayer();
Axis axisY = gl.YAxis;
Tree tr;

// Show major grid
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; // Solid
tr.Root.Grids.HorizontalMajorGrids.Width.dVal = 1;

// Show minor grid
trProperty = tr.Root.Grids.HorizontalMinorGrids.AddNode("Show");
trProperty.nVal = 1;
tr.Root.Grids.HorizontalMinorGrids.Color.nVal = SYSCOLOR_GREEN; //
Green
tr.Root.Grids.HorizontalMinorGrids.Style.nVal = 2; // Dot
tr.Root.Grids.HorizontalMinorGrids.Width.dVal = 0.3;

if(0 == axisY.UpdateThemeIDs(tr.Root) )
{
        bool bRet = axisY.ApplyFormat(tr, true, true);
}
```

6.1.8 Setting Axis Scale

This example shows how to set scale parameters, increment, type and so on.

```
GraphLayer gl = Project.ActiveLayer();
Axis axesX = gl.XAxis;

axesX.Scale.From.dVal = 0;
axesX.Scale.To.dVal = 1;
axesX.Scale.IncrementBy.dVal = 0.2;
axesX.Scale.Type.nVal = 0;// Linear
axesX.Scale.Rescale.nVal = 0; // Rescake type
axesX.Scale.RescaleMargin.dVal = 8; // precent 8
```

6.1.9 Getting Axis Format

```
GraphLayer gl = Project.ActiveLayer();
Axis axisX = gl.XAxis;

// Get all axis format settings to tree
Tree tr;
tr = axisX.GetFormat(FPB_ALL, FOB_ALL, true, true);
out_tree(tr);
```

6.1.10 Setting Axis Label

An axis label is an ordinary text object and is accessed in Origin C using the GraphObject class. On a default graph the X axis is named XB and the Y axis is named YL. The following code shows how to access the X and Y axis labels and assumes a default graph is the active page.

```
GraphLayer gl = Project.ActiveLayer();    // Get active graph layer

GraphObject grXL = gl.GraphObjects("XB"); // Get X axis label
GraphObject grYL = gl.GraphObjects("YL"); // Get Y axis label
```

Now that we have access to the axis labels we can change their values. The following code sets the X axis label directly and sets the Y axis label indirectly by linking it to a LabTalk string variable. Linking to a LabTalk variable requires the label's Programming Control option "Link to variables" to be turned on. This option is on by default.

```
grXL.Text = "My New X Asis Label";

LT_set_str("abc$", "My String Variable");
grYL.Text = "%(abc$)";
```

To make sure the label changes appear, it may be necessary to refresh the graph page. With our GraphLayer object we can refresh the page with the following code.

```
gl.GetPage().Refresh();
```

6.1.11 Show Top Axis

This example shows how to show X top axes.

```
// Show axes and ticks
Tree tr;
TreeNode trProperty = tr.Root.Ticks.TopTicks.AddNode("Show");
trProperty.nVal = 1;

// Show tick labels
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);
}
```

6.1.12 Customizing Axis Ticks

This example shows how to set the format in the Axis dialog -> Title & Format tab.

```
GraphLayer gl = Project.ActiveLayer();
Axis axesX = gl.XAxis;

Tree tr;
// Set ticks color as Auto, depend on the color of data plot
```

```
tr.Root.Ticks.BottomTicks.Color.nVal = INDEX_COLOR_AUTOMATIC;
tr.Root.Ticks.BottomTicks.Width.dVal = 3;
tr.Root.Ticks.BottomTicks.Major.nVal = 0; // 0: In and Out
tr.Root.Ticks.BottomTicks.Minor.nVal = 2; // 2: Out
tr.Root.Ticks.BottomTicks.Style.nVal = 0; // Solid

if(0 == axesX.UpdateThemeIDs(tr.Root) )
    bool bRet = axesX.ApplyFormat(tr, true, true);
```

6.1.13Customizing Tick Labels

This example shows how to set tick labels with custom positions. It performs the same action as going in the Axis dialog Custom Tick Labels tab.

```
GraphLayer gl = Project.ActiveLayer();
Axis axesX = gl.XAxis;

Tree tr;
// Show axes begin and end as scale value
tr.Root.Labels.BottomLabels.Custom.Begin.Type.nVal = 2;
tr.Root.Labels.BottomLabels.Custom.End.Type.nVal = 2;

// Set special point as Manual type with the special value and text.
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);
}
```

6.2 Adding Data Plots

Plots or **Data plots** are representations of your data within a graph layer. Each graph layer may contain one or more plots.

6.2.1 2D Plot (XY, YErr, Bar/Column)

Plot XY Scatter

The following code shows how to construct an XYYErr data range from the active worksheet, and then plot the data range in a newly created graph.

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

// The range name must be X, Y, Z or ED(for YErr) to make sense.
DataRange dr;
dr.Add(wks, 0, "X"); // 1st column for X data
dr.Add(wks, 1, "Y"); // 2nd column for Y data
dr.Add(wks, 2, "ED"); // Optional, 3th column for Y Error data
```

```
// Create a graph window
GraphPage gp;
gp.Create("Origin");
GraphLayer gl = gp.Layers(); // Get active layer

// Plot XY data range as scatter
// IDM_PLOT_SCATTER is plot type id, see other types plot id in
oPlotIDs.h file.
int nPlotIndex = gl.AddPlot(dr, IDM_PLOT_SCATTER);
// Returns plot index (offset is 0), else return -1 for error
if( nPlotIndex >= 0 )
{
        gl.Rescale(); // Rescale axes to show all data points
}
```

Attach YErr Plot

Attach YErr data to an existing XY data plot.

```
GraphLayer gl = Project.ActiveLayer();
DataPlot dp = gl.DataPlots(-1);       // Get active data plot

// Get Y Error column
WorksheetPage wksPage("Book1");
Worksheet wks = wksPage.Layers();
Column  colErrBar(wks, 2);

// Plot Y Error column to the active data plot
Curve   crv(dp);
int             nErrPlotIndex = gl.AddErrBar(crv, colErrBar);
out_int("nErrPlotIndex = ", nErrPlotIndex);
```

Bar/Column Plot

```
// before running make sure the active window is worksheet
Worksheet wks = Project.ActiveLayer();
DataRange dr;
dr.Add(wks, 1, "Y"); // Construct data range with one column

GraphPage gp;
gp.Create("BAR"); // Create graph with the specified template
GraphLayer gl = gp.Layers(-1); // Get active graph layer

int index = gl.AddPlot(dr, IDM_PLOT_BAR);
if( index >= 0 )
{
        out_str("Plot bar");
        gl.Rescale();
}
```

6.2.2 3D Plot

Plot a 3D surface from a matrix on a graph window.

```
// Prepare matrix data
MatrixLayer ml;
```

```
string strFile = GetAppPath(true) + "Samples\\Matrix Conversion and
Gridding\\
2D Gaussian.ogm";
ml.Open(strFile);
MatrixObject mo = ml.MatrixObjects(0);

// Create graph page with template
GraphPage gp;
gp.Create("CMAP");
GraphLayer gl = gp.Layers(0);

// Plot 3D surface
int nPlotIndex = gl.AddPlot(mo, IDM_PLOT_SURFACE_COLORMAP);
if(0 == nPlotIndex)
{
        gl.Rescale();
        printf("3D Surface plotted successfully\n");
}
```

6.2.3 Contour Plot

Plot XYZ Contour

```
// Before running, make sure there are XYZ columns with data in the
active
// worksheet window. Or you can import \Samples\Matrix Conversion and
Gridding\
// XYZ Random Gaussian.dat into worksheet.
Worksheet wks = Project.ActiveLayer();
DataRange dr;
dr.Add(wks, 0, "X");
dr.Add(wks, 1, "Y");
dr.Add(wks, 2, "Z");

// Create graph with template
GraphPage gp;
gp.Create("TriContour");
GraphLayer gl = gp.Layers();

// Plot XYZ contour with type id
int nPlot = gl.AddPlot(dr, IDM_PLOT_TRI_CONTOUR);
if( nPlot >= 0 )
{
        gl.Rescale();
        printf("XYZ contour plotted successfully\n");
}
```

Plot Color Fill Contour

```
MatrixLayer ml = Project.ActiveLayer();
MatrixObject mo = ml.MatrixObjects(0);

// Create graph window with template
GraphPage gp;
gp.Create("contour");
GraphLayer gl = gp.Layers();
```

```
int nPlot = gl.AddPlot(mo, IDM_PLOT_CONTOUR);
if( nPlot >= 0 )
{
        gl.Rescale();
}
```

6.2.4 Image Plot

```
MatrixLayer ml = Project.ActiveLayer();
MatrixObject mo = ml.MatrixObjects(0);

// Create graph window with template
GraphPage gp;
gp.Create("image");
GraphLayer gl = gp.Layers();

int nPlot = gl.AddPlot(mo, IDM_PLOT_MATRIX_IMAGE);
if( nPlot >= 0 )
{
        gl.Rescale();
}
```

6.2.5 Multi-Axes

The following example code shows how to show/hide and set format on the four axes - left, bottom, right, and top in one graph layer.

```
#include <..\Originlab\graph_utils.h> // needed for AXIS_*
GraphLayer gl = Project.ActiveLayer();

// Show all axes and labels. 0 or 1, 1 for show.
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;

// Show axis titles of left and bottom axes. 0 or 1, 1 for show.
vnTitles[AXIS_BOTTOM] = 1;
vnTitles[AXIS_LEFT] = 1;
vnTitles[AXIS_TOP] = 0;
vnTitles[AXIS_RIGHT] = 0;

// Set the major tick and minor tick of all axes as IN format
// See other TICK_* items in graph_utils.h.
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);
```

6.2.6 Multi-Panels (Multi-Layer, with Shared X-Axis)

The following example shows how to construct multiple graph layers in one graph page, all layers sharing the x axis in one layer, then plot XY data sets one by one from a worksheet to each graph layer.
Before compiling the following codes, you need to run this command to build the graph_utils.c file to your current workspace.

```
run.LoadOC(Originlab\graph_utils.c, 16);
```

Compile the following Origin C code. Before running, make sure there is a workbook named Book1, and it has one X column and at least two Y columns.

```
#include <..\Originlab\graph_utils.h> // needed for page_add_layer
function
// Construct data range from Book1
WorksheetPage wksPage("Book1");
Worksheet wks = wksPage.Layers(0); // get the first worksheet in Book1
DataRange dr;
dr.Add(wks, 0, "X"); // 1st column as X data
dr.Add(wks, 1, "Y", -1); // 2nd column to last one for Y data

// Get the number of Y
DWORD dwRules = DRR_GET_DEPENDENT | DRR_NO_FACTORS;
int nNumYs = dr.GetNumData(dwRules);

// Add more layers with right Axis and link to the 1st layer
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);
}

// Loop and add plot from each XY data range to graph layer
foreach(GraphLayer gl in gp.Layers)
{
        int nLayerIndex = gl.GetIndex();

        // Get the sub XY range from dr
        DataRange drOne;
        dr.GetSubRange(drOne, dwRules, nLayerIndex);

        // Plot one XY range to graph layer
        int nPlot = gl.AddPlot(drOne, IDM_PLOT_LINE);
        if( nPlot >= 0 )
        {
                DataPlot dp = gl.DataPlots(nPlot);
                dp.SetColor(nLayerIndex); // Set data plot as different
color

                // Set the ticks and ticklabels of right Y axis auto
color
```

```
        gl.YAxis.AxisObjects(AXISOBJPOS_AXIS_SECOND).RightTicks.Color.nV
al =
        gl.YAxis.AxisObjects(AXISOBJPOS_LABEL_SECOND).RightLabels.Color.
nVal =
                INDEX_COLOR_AUTOMATIC;

                gl.Rescale();
        }
}
```

6.3 Customizing Data Plots

6.3.1 Adding Data Marker

Origin C supports the following methods for customizing data markers.

- **DataPlot::AddDataMarkers** to add a data marker on the data plot to select a sub range
- **DataPlot::SetDataMarkers** to change the position of the present data marker
- **DataPlot::GetDataMarkers** to get all existing data plots
- **DataPlot::RemoveDataMarker** to remove the specified data marker.

The following code shows how to add two data markers to the active graph window.

```
GraphLayer gl = Project.ActiveLayer();
DataPlot dp = gl.DataPlots();

// the indices of the data markers
vector<int> vnBegin = {0, 9};
vector<int> vnEnd = {4, 14};

// to add two data markers
int nRet = dp.AddDataMarkers(vnBegin, vnEnd);
if( 0 == nRet )
{
        out_str("Add data marker successfully.");
}
```

The code below shows how to change the position of the present data marker.

```
GraphLayer gl = Project.ActiveLayer();
DataPlot dp = gl.DataPlots();

// the indices of the data markers
vector<int> vnBegin = {11, 2};
vector<int> vnEnd = {19, 5};
vector<int> vnIndices = {1, 0};

// to add two data markers
int nRet = dp.SetDataMarkers(vnBegin, vnEnd, vnIndices);
```

```
if( 0 == nRet )
{
        out_str("Set data marker successfully.");
        gl.GetPage().Refresh();
}
```

6.3.2 Setting Color

The following code shows how to set the color of the data plot.

```
GraphLayer gl = Project.ActiveLayer();
DataPlot dp = gl.DataPlots(0);

bool bRepaint = true;
dp.SetColor(SYSCOLOR_GREEN, bRepaint);
```

6.3.3 Getting Format Tree

OriginObject::GetFormat and *OriginObject::ApplyFormat* are used to get and set Origin object formats. The following getting, setting and copying format mechanisms can be used for all Origin objects whose classes derive from the OriginObject base class (see Reference: Class Hierarchy). For example, the Origin objects can be objects of the *DataPlot* class, *Worksheet* class, *WorksheetPage* class, *MatrixLayer* class, *MatrixPage* class, *GraphLayer* class, or *GraphPage* class.
The *DataPlot* class derives from the *DataObjectBase* class, and the *DataObjectBase* class derives from the *OriginObject* class, so we can call *DataPlot::GetFormat* to get the format tree structure.
There are two ways to see the format tree structure via the following code.

- Set a break point on the *GetFormat* line in the following code, activate one data plot, run the code, press F10 (Step Over) to execute the *GetFormat* line, and see the details of the format tree in the Code Builder Local Variables Window **tr** variable. (press Alt+4 to open/hide the Local Variables window).
- Use the last line, *out_tree(tr);*, to print out the format tree.

```
GraphLayer gl = Project.ActiveLayer();
DataPlot dp = gl.DataPlots(-1); // Get the active data plot

// Different plot types(for example, Line, Box Chart...) have
// different structure in the format tree.
Tree tr;

// Get the format tree to see details of the tree structure.
tr = dp.GetFormat(FPB_ALL, FOB_ALL, true, true);

out_tree(tr); // print out the format tree.
```

6.3.4 Setting Format on Line Plot

```
GraphLayer gl = Project.ActiveLayer();
DataPlot dp = gl.DataPlots(-1); // Get the active data plot

// Set format on a line plot
// Note: See the previous section to get the structure of format tree
Tree tr;
tr.Root.Line.Connect.nVal = 2; // 2 for 2 point segment
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);
}
```

6.3.5 Copying Format from One Data Plot to Another

Copying Format via Theme File

Getting and saving a format tree from a data plot into a theme file, then loading the theme file to a tree and applying the format tree to another data plot.

```
// Save plot settings from Graph1 to a theme file
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);

// Load plot settings from a theme file to a tree, and apply format
from
// tree to data plot object.
GraphPage gpDest("Graph2");
GraphLayer glDest = gpDest.Layers(0);
DataPlot dpDest = glDest.DataPlots(0);

Tree tr2;
tr2.Load(strTheme);
dpDest.ApplyFormat(tr2, true, true);
```

Copying Format via Tree

Getting plot settings from one data plot to a tree, then apply settings from this tree to another data plot object.

```
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);

// Get format from source data plot
Tree tr;
tr = dpSource.GetFormat(FPB_ALL, FOB_ALL, true, true);

// Apply format to another data plot
dpDest.ApplyFormat(tr, true, true);
```

6.3.6 Setting Format on Scatter Plot

```
GraphLayer gl = Project.ActiveLayer();
DataPlot dp = gl.DataPlots(-1); // Get the active data plot

// Set symbol format
Tree tr;
tr.Root.Symbol.Size.nVal = 12; // Size of symbol
tr.Root.Symbol.Shape.nVal = 1; // Circle
tr.Root.Symbol.Interior.nVal = 1; // Interior type
tr.Root.Symbol.EdgeColor.nVal = SYSCOLOR_RED;
tr.Root.Symbol.FillColor.nVal = SYSCOLOR_BLUE;

// Show vertical droplines
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);
}
```

6.3.7 Setting Format on Grouped Line + Symbol Plots

Use Origin C to set the format for grouped plots. The same action can be completed by going into the Plot Details dialog, under the Group tab. The formats included Line Color, Symbol Type, Symbol Interior, and Line Style.
The following example shows how to set format on Line and Symbol plots. This group is assumed to contain 4 data plots.

```
GraphLayer gl = Project.ActiveLayer();
GroupPlot gplot = gl.Groups(0); // Get the first group in layer

// the Nester is an array of types of objects to do nested cycling in
the group
// four types of setting to do nested cycling in the group
vector<int> vNester(3);
vNester[0] = 0;  // cycling line color in the group
vNester[1] = 3;  // cycling symbol type in the group
vNester[2] = 8;  // cycling symbol interior in the group
gplot.Increment.Nester.nVals = vNester;  // set Nester of the grouped
plot

// Put format settings to vector for 4 plots
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;  // set line color to
theme tree
tr.Root.Increment.Shape.nVals = vSymbolShape;  // set symbol shape to
theme tree
// set symbol interior to theme tree
tr.Root.Increment.SymbolInterior.nVals = vSymbolInterior;
```

```
if(0 == gplot.UpdateThemeIDs(tr.Root) )
{
        bool bb = gplot.ApplyFormat(tr, true, true);    // apply theme
tree
}
```

6.3.8 Setting Colormap Settings

DataPlot class has two overloaded methods to set colormap.

- DataPlot::SetColormap(const vector<double> & vz, BOOL bLogScale = FALSE) is just used to set Z level and scale type (log type or not). The values in **vz** argument are Z values.
- DataPlot::SetColormap(TreeNode& trColormap) is used to set all colormap settings, for example, Z values, colors, line format and text label format.

This example shows how to set up colormap Z levels on a Contour graph.

```
GraphLayer gl = Project.ActiveLayer();
DataPlot dp = gl.DataPlots(0);

// Get original colormap Z levels
vector vZs;
BOOL bLogScale = FALSE;
BOOL bRet = dp.GetColormap(vZs, bLogScale);
int nLevels = vZs.GetSize();

// Decrease Z levels vector and set back to DataPlot
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);
```

The following example shows how to set up colormap Z levels with log10 scale type.

```
bool plot_matrix(LPCSTR lpsczMatPage, LPCSTR lpcszGraphTemplate =
"contour"
                , int nPlotID = IDM_PLOT_CONTOUR)
{
        // Get the active matrix object from the specific matrix page
        MatrixPage matPage = Project.MatrixPages(lpsczMatPage);
        if( !matPage )
        {
                out_str("Invalid matrix page");
                return false;
        }
        // get the active sheet in this matrix page
        MatrixLayer ml = matPage.Layers(-1);
        // get the active matrix object in matrixsheet
        MatrixObject mobj = ml.MatrixObjects(-1);

        // Create hidden graph page with template and add plot
        // Create as hidden to avoid unneeded drawing
        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(); // rescale x y axes


        // Construct Z levels vector
        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;

        // Setup z levels in percent, not real z values.
        // First value must be 0 and last value must be < 100
        vLevels = 100*(vLevels - min)/(max - min);

        Tree tr;
        tr.ColorMap.Details.Levels.dVals = vLevels;
        tr.ColorMap.ScaleType.nVal = 1; // 1 for 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(); // show it when all it ready

        return true;
}
```

Call the above *plot_matrix* function with *coutour* template and *IDM_PLOT_CONTOUR* plot id to plot contour graph and then set colormap on it.

```
void plot_contour_ex(LPCSTR lpcszMatPage)
{
    plot_matrix(lpcszMatPage, "contour", IDM_PLOT_CONTOUR);
}
```

Call the above *plot_matrix* function with *image* template and *IDM_PLOT_MATRIX_IMAGE* plot id to plot image graph and then set colormap on it.

```
void plot_image_ex(LPCSTR lpcszMatPage)
{
    plot_matrix(lpcszMatPage, "image", IDM_PLOT_MATRIX_IMAGE);
}
```

The following example shows how to remove fill color, and set up line color, style, width and text labels on a Contour graph.

```
GraphLayer gl = Project.ActiveLayer();
DataPlot dp = gl.DataPlots(0);

Tree tr;
dp.GetColormap(tr);

// Remove fill color
tr.ColorFillControl.nVal = 0;

// Set line color
vector<int> vnLineColors;
vnLineColors = tr.Details.LineColors.nVals;
int nLevels = vnLineColors.GetSize();
vnLineColors.Data(1, nLevels, 1);
tr.Details.LineColors.nVals = vnLineColors;

// Set line style as Dash for all lines
vector<int> vnLineStyles(vnLineColors.GetSize());
vnLineStyles = 1;
tr.Details.LineStyles.nVals = vnLineStyles;

// Set line width for all lines
vector vdLineWidths(vnLineColors.GetSize());
vdLineWidths = 3;
tr.Details.LineWidths.dVals = vdLineWidths;

// Show/hide labels, show all except that the first two.
vector<int> vnLabels(vnLineColors.GetSize());
vnLabels = 1;
vnLabels[0] = 0;
vnLabels[1] = 0;
tr.Details.Labels.nVals = vnLabels;

// Set back settings to graph
dp.SetColormap(tr);
```

This example shows how to set the format(i.e. color, size, bold, italic) of the text labels on a Contour graph.

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

```
DataPlot dp = gl.DataPlots(0);

// Get all properties of the related objects of the colormap data plot
Tree tr;
tr = dp.GetFormat(FPB_ALL, FOB_ALL, true, true);

// Show all labels
vector<int> vnLabels;
vnLabels = tr.Root.ColorMap.Details.Labels.nVals;
vnLabels = 1;// 0 to hide, 1 to show
tr.Root.ColorMap.Details.Labels.nVals = vnLabels;

// Set the numeric format for labels
tr.Root.NumericFormats.Format.nVal = 0; // Decimal
tr.Root.NumericFormats.DigitsControl.nVal = 0;
tr.Root.NumericFormats.SignificantDigits.nVal = 5;//DecimalPlaces
tr.Root.NumericFormats.Prefix.strVal = "_";
tr.Root.NumericFormats.Suffix.strVal = "Label";
tr.Root.NumericFormats.MinArea.nVal = 5; // Labeling Criteria - Min
Area(%)

// Set text format for labels
tr.Root.Labels.Color.nVal = SYSCOLOR_BLUE;
//FontFaceIndex_to_DWORD is used to convert font from GUI index to
DWORD real value
tr.Root.Labels.Face.nVal = FontFaceIndex_to_DWORD(2);// choose the 3rd
font in GUI
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);
```

6.4 Managing Layers

6.4.1 Creating a Panel Plot

Creating a 6 Panel Graph

The following example will create a new graph window with 6 layers, arranged as 2 columns and 3 rows. This function can be run independent of what window is active.

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

while(gp.Layers.Count() < 6)
{
        gp.AddLayer();
}

graph_arrange_layers(gp, 3, 2);
```

Creating and Plotting into a 6 Panel Graph

The following example will import some data into a new workbook, create a new graph window with 6 layers, arranged as 2 columns and 3 rows, and loop through each layer (panel), plotting the imported data.

```
// Import data file to worksheet
ASCIMP          ai;
Worksheet       wks;
string  strDataFile = GetOpenBox(FDLOG_ASCII, GetAppPath(true));
if(AscImpReadFileStruct(strDataFile,&ai) == 0)
{
        wks.Create("Origin");
        wks.ImportASCII(strDataFile, ai);
}

// Add XY data from worksheet to graph each layers
GraphPage gp("Graph1"); // the graph has the 3x2 panel layers created
above
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++;
}
```

6.4.2 Adding Layers to a Graph Window

The following example will add an independent right Y axis scale. A new layer is added, displaying only the right Y axis. It is linked in dimension and the X axis is linked to the current active layer at the time the layer is added. The new added layer becomes the active layer.
Before compiling the following codes, you need to add graph_utils.c to your current workspace. Run Labtalk command "Run.LoadOC(Originlab\graph_utils.c)" to do this.

```
#include <..\Originlab\graph_utils.h>// Needed for page_add_layer
function

GraphLayer gl = Project.ActiveLayer();
GraphPage gp = gl.GetPage();

bool bBottom = false, bLeft = false, bTop = false, bRight = true;
int nLinkTo = gl.GetIndex(); // New added layer link to the active
layer
bool bActivateNewLayer = true;

int     nLayerIndex = page_add_layer(gp, bBottom, bLeft, bTop, bRight,
ADD_LAYER_INIT_SIZE_POS_SAME_AS_PREVIOUS, bActivateNewLayer, nLinkTo);
```

6.4.3 Arranging the Layers

The following example will arrange the existing layers on the active graph into two rows by three columns. If the active graph does not already have 6 layers, it will not add any new layers. It arranges only the layers that exist.

```
GraphLayer gl = Project.ActiveLayer();
GraphPage gp = gl.GetPage();

int nRows = 3, nCols = 2;
graph_arrange_layers(gp, nRows, nCols);
```

6.4.4 Moving a Layer

The following example will left align all layers in the active graph window, setting their position to be 15% from the left-hand side of the page.

```
GraphLayer gl = Project.ActiveLayer();
GraphPage gp = gl.GetPage();

int nRows = gp.Layers.Count();
int nCols = 1;

stLayersGridFormat stFormat;
stFormat.nXGap = 0; // the X direction gap of layers
stFormat.nYGap = 5; // the Y direction gap of layers
stFormat.nLeftMg = 15; // left margin
stFormat.nRightMg = 10;
stFormat.nTopMg = 10;
stFormat.nBottomMg = 10;

page_arrange_layers(gp, nRows, nCols, &stFormat);
```

6.4.5 Resizing a Layer

The following example will resize the current layer to reduce the width and height to half of the original size.
Before compiling the following codes, you need to add graph_utils.c to your current workspace. Run Labtalk command "Run.LoadOC(Originlab\graph_utils.c)" to do this.

```
#include <..\Originlab\graph_utils.h> // Needed for layer_set_size
function
GraphLayer gl = Project.ActiveLayer();

// get the original size of graph layer
double dWidth, dHeight;
layer_get_size(gl, dWidth, dHeight);

// resize layer
dWidth /= 2;
dHeight /= 2;
layer_set_size(gl, dWidth, dHeight);
```

6.4.6 Swap two Layers

The following example will swap the position on the page of layers indexed 1 and 2. Before compiling the following codes, you need to add graph_utils.c to your current workspace. Run Labtalk command "Run.LoadOC(Originlab\graph_utils.c)" to do this.

```
#include <..\Originlab\graph_utils.h> // Needed for layer_swap_position
function
GraphPage gp("Graph1");
GraphLayer gl1 = gp.Layers(0);
GraphLayer gl2 = gp.Layers(1);

layer_swap_position(gl1, gl2);
```

The following example will swap the position on the page of layers named Layer1 and Layer2.

```
GraphPage gp("Graph1");
GraphLayer gl1 = gp.Layers("Layer1");
GraphLayer gl2 = gp.Layers("Layer2");

layer_swap_position(gl1, gl2);
```

6.4.7 Aligning Layers

The following example will bottom align layer 2 with layer 1 in the active graph window.
Before compiling the following codes, you need to add graph_utils.c to your current workspace. Run Labtalk command "Run.LoadOC(Originlab\graph_utils.c)" to do this.

```
#include <..\Originlab\graph_utils.h> // Needed for layer_aligns
function
// Get the active graph page
GraphLayer gl = Project.ActiveLayer();
GraphPage gp = gl.GetPage();

GraphLayer gl1 = gp.Layers(0);
GraphLayer gl2 = gp.Layers(1);

// Bottom align layer 2 with layer 1
layer_aligns(gl1, gl2, POS_BOTTOM);
```

6.4.8 Linking Layers

The following example will link all X axes in all layers in the active graph to the X axis of layer 1. The Units will be set to a % of Linked Layer.
Before compiling the following codes, you need to add graph_utils.c to your current workspace. Run Labtalk command "Run.LoadOC(Originlab\graph_utils.c)" to do this.

```
#include <..\Originlab\graph_utils.h> // Needed for layer_set_link
function
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; // Set layer unit as % of linked layer
        if( glOne != gl1 )
                layer_set_link(glOne, gl1.GetIndex(), LINK_STRAIGHT,
LINK_NONE, &nUnit);
}
```

6.4.9 Setting Layer Unit

```
int nUnit = M_PIXEL;
GraphLayer gl = Project.ActiveLayer();

// Get the current position
double dPos[TOTAL_POS];
gl.GetPosition(dPos);

// Convert position to the specified unit
gl.UnitsConvert(nUnit, dPos);

// Set position with unit
gl.SetPosition(dPos, nUnit);
```

6.5 Creating and Accessing Graphical Objects

6.5.1 Creating Graphical Object

Add a Graphical Object, for example: text, or a rectangle or line.
The following example shows how to add a rectangle to the active graph. For other Graph object types see GROT_* (for example: GROT_TEXT, GROT_LINE, GROT_POLYGON) in the oc_const.h file.

```
GraphLayer gl = Project.ActiveLayer();
string strName = "MyRect";
GraphObject goRect = gl.CreateGraphObject(GROT_RECT, strName);
```

Add a text label on the current graph window:

```
GraphLayer gl = Project.ActiveLayer();
GraphObject go = gl.CreateGraphObject(GROT_TEXT, "MyText");
go.Text = "This is a test";
```

The example below shows how to add an arrow to a graph. The object type of an arrow is GROT_LINE, the same type as a line. And for both lines and arrows, the number of data points required is 2.

```
GraphPage gp;
gp.Create();
GraphLayer gl = gp.Layers();

string strName = "MyArrow"; // the name of the graph object
GraphObject go = gl.CreateGraphObject(GROT_LINE, strName);

go.Attach = 2; // change attach mode to Layer and Scale
```

```
Tree tr;
tr.Root.Dimension.Units.nVal = 5; // Set unit as Scale

// Set position by scale value
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);
}
```

The example below shows how to add a curved arrow to a graph. For a curved arrow, the number of data points required is 4.

```
GraphPage gp;
gp.Create();
GraphLayer gl = gp.Layers();

string strName = "MyArrow"; // the name of the graph object
GraphObject go = gl.CreateGraphObject(GROT_LINE4, strName);

go.Attach = 2; // change attach mode to Layer and Scale

Tree tr;
tr.Root.Dimension.Units.nVal = 5; // Set unit as Scale

// Set position by scale value
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);
}
```

6.5.2 Setting Properties

Set Properties for a Graphical Object, for example, text font, color, line width.

```
// Set color and font for graph object
GraphLayer gl = Project.ActiveLayer();
GraphObject goText = gl.GraphObjects("Text");
goText.Text = "This is a test";
goText.Attach = 2; // Attach to layer scale
```

```
Tree tr;
tr.Root.Color.nVal = SYSCOLOR_RED; // the color of text
tr.Root.Font.Bold.nVal = 1;
tr.Root.Font.Italic.nVal = 1;
tr.Root.Font.Underline.nVal = 1;
tr.Root.Font.Size.nVal = 30; // font size of text

if( 0 == goText.UpdateThemeIDs(tr.Root) )
{
        bool bRet = goText.ApplyFormat(tr, true, true);
}
```

6.5.3 Setting Position and Size

```
GraphLayer gl = Project.ActiveLayer();
GraphObject go = gl.GraphObjects("Rect");
go.Attach = 2; // Attach to layer scale

// Move text object to the layer left top
Tree tr;
tr.Root.Dimension.Units.nVal = UNITS_SCALE;
tr.Root.Dimension.Left.dVal = gl.X.From; // Left
tr.Root.Dimension.Top.dVal = gl.Y.To/2; // Top
tr.Root.Dimension.Width.dVal = (gl.X.To - gl.X.From)/2; // Width
tr.Root.Dimension.Height.dVal = (gl.Y.To - gl.Y.From)/2; // Height

if( 0 == go.UpdateThemeIDs(tr.Root) )
{
        bool bRet = go.ApplyFormat(tr, true, true);
}
```

6.5.4 Updating Attach Property

The attach property has 3 choices, Page, Layer Frame, and Layer Scale.

```
// Attach graph object to the different object:
// 0 for layer, when move layer, graph object will be moved together;
// 1 for page, when move layer, not effect on graph object;
// 2 for layer scale, when change the scale, the position of graph
object
// will be changed according.
go.Attach = 2;
```

6.5.5 Getting and Setting Disable Property

```
// To check disable properties, for example, movable, selectable.
Tree tr;
tr = go.GetFormat(FPB_OTHER, FOB_ALL, true, true);
DWORD   dwStats = tr.Root.States.nVal;

// To check vertical and horizontal movement.
// More property bits, see GOC_* in oc_const.h file.
if( (dwStats & GOC_NO_VMOVE) && (dwStats & GOC_NO_HMOVE) )
{
        out_str("This graph object cannot be move");
}
```

6.5.6 Programming Control

```
// 1. Add a line
GraphLayer gl = Project.ActiveLayer();
GraphObject go = gl.CreateGraphObject(GROT_LINE);
go.Attach = 2; // Set attach mode to layer scale
go.X = 5; // Set init position to X = 5

// 2. Set line properties
Tree tr;
tr.Root.Direction.nVal = 2; // 1 for Horizontal, 2 for vertical
tr.Root.Span.nVal = 1; // Span to layer
tr.Root.Color.nVal = SYSCOLOR_RED; // Line color

if( 0 == go.UpdateThemeIDs(tr.Root) )
{
        go.ApplyFormat(tr, true, true);
}

// 3. Set event mode and LT script.
// Move line will print out line position, x scale value.
Tree trEvent;
trEvent.Root.Event.nVal = GRCT_MOVE;// More other bits, see GRCT_* in
oc_const.h
trEvent.Root.Script.strVal = "type -a $(this.X)";

if( 0 == go.UpdateThemeIDs(trEvent.Root) )
{
        go.ApplyFormat(trEvent, true, true);
}
```

6.5.7 Updating Legend

A legend is a graphical object named "Legend" on a graph window. After adding/removing data plots, we can use the **legend_update** function to refresh the legend according to the current data plots.

```

// Simple usage here, just used to refresh legend.
// Search this function in OriginC help to see the description of other
arguments
// for more usages.
legend_update(gl); // gl is a GraphLayer object
```

6.5.8 Adding Table Object on Graph

```
// 1. Create the worksheet with Table template
Worksheet wks;
wks.Create("Table", CREATE_HIDDEN);
WorksheetPage wksPage = wks.GetPage();

// 2. Set table size and fill in text
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. Add table as link to graph
GraphLayer gl = Project.ActiveLayer();
GraphObject grTable = gl.CreateLinkTable(wksPage.GetName(), wks);
```

7 Working with Data

7.1 Numeric Data

This section gives examples of working with numeric data in Origin C. Numeric data can be stored in variables of the following data types:

1. double
2. integer
3. vector
4. matrix

Numeric data and strings can be stored in the nodes of a tree, provided the nodes have one of the data types above.

7.1.1 Missing Values

As important as numeric data is, it is also important to be able to represent missing data. Origin C defines the NANUM macro for comparing and assigning values to missing data. Missing values are only supported with the double data type.

```
double d = NANUM;
if( NANUM == d )
    out_str("The value is a missing value.");
```

Origin C also provides the **is_missing_value** function for testing if a value is a missing value.

```
if( is_missing_value(d) )
    out_str("The value is a missing value.");
```

7.1.2 Precision and Comparison

In the following example code, the *prec* and *round* functions are used to control the precision of double type numeric data. The *is_equal* function is used to compare two pieces of double type numeric data.

```
double dVal = PI; // PI defined as 3.1415926535897932384626

// convert the double value to have 6 significant digits
int nSignificantDigits = 6;
printf("%f\n", prec(dVal, nSignificantDigits));

// force the double value to only have two decimal digits
uint nDecimalPlaces = 2;
double dd = round(dVal, nDecimalPlaces);
printf("%f\n", dd);

// compare two double values
if( is_equal(dd, 3.14) )
{
       out_str("equal\n");
}
```

```
else
{
        out_str("not equal\n");
}
```

7.1.3 Convert Numeric to String

```
// assign int type numeric to string
string str = 10;
out_str(str);

int nn = 0;
str = nn;
out_str(str);

// convert double type numeric to string
double dd = PI;
str = ftoa(dd, "*"); // Use "*" for Origin's global setting in Options
dialog
out_str(str);

str = ftoa(dd, "*8"); // Use "*8" for 8 significant
out_str(str);
```

7.1.4 Vector

```
// One-Dimensional array with basic data type, for example, double,
int, string,
// complex.
vector vx, vy;

int nMax = 10;
vx.Data(1, nMax, 1); // assign value to vx from 1 to 10 with increment
1
vy.SetSize(nMax); // set size(10) to vy

for(int nn = 0; nn < nMax; nn++)
{
        vy[nn] = rnd(); // assign random data to each item in vy
        printf("index = %d, x = %g, y = %g\n", nn+1, vx[nn], vy[nn]);
}
// Access the data in a worksheet window
Worksheet wks = Project.ActiveLayer();
Column col(wks, 0);

vector& vec = col.GetDataObject();
vec = vec * 0.1; // Multiply 0.1 by each piece of data in vec

vec = sin(vec);// Find the sine of each piece of data in vec
```

7.1.5 Matrix

```
// Two-Dimensional array with basic data type, for example, double,
int, complex,
// but not string.
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"); // new line
}
// Access the data in matrix window
MatrixLayer ml = Project.ActiveLayer();
MatrixObject mo = ml.MatrixObjects(0);

matrix& mat = mo.GetDataObject();
mat = mat + 0.1; // Add 0.1 for the each data in matrix
```

7.1.6 TreeNode

The Origin C TreeNode class provides several methods for constructing multi-level trees, traversing trees and accessing the value/attributes of tree nodes.

```
Tree tr;

// Access the value of a tree node
TreeNode trName = tr.AddNode("Name");
trName.strVal = "Jane";

tr.UserID.nVal = 10;

vector<string> vsBooks = {"C++", "MFC"};
tr.Books.strVals = vsBooks;

out_tree(tr); // output tree
```

7.1.7 Complex

```
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); // output cc = 2.500000+0.220000i
// Access complex dataset
Worksheet wks = Project.ActiveLayer();
Column col(wks, 1);
if( FSI_COMPLEX == col.GetInternalDataType() )
{
        vector<complex>& vcc = col.GetDataObject();
        vcc[0] = 0.5 + 3.6i;
}
// Access complex matrix
MatrixLayer ml = Project.ActiveLayer();
MatrixObject mo = ml.MatrixObjects();

if( FSI_COMPLEX == mo.GetInternalDataType() )
{
        matrix<complex>& mat = mo.GetDataObject();
        mat[0][0] = 1 + 2.5i;
}
```

7.1.8 DataRange

The **DataRange** class is a versatile mechanism to get and put data in a Worksheet, Matrix or Graph window.

Data Range in Worksheet

For a Worksheet, a data range can be specified by column/row index as one column, one row, any sub block range, one cell or entire Worksheet.

```
// Construct a data range on the active worksheet, all columns and rows
// from 1st row to 5th row.
Worksheet wks = Project.ActiveLayer();
int r1 = 0, c1 = 0, r2 = 4, c2 = -1;

DataRange dr;
// range name should be make sense, for example, "X", "Y",
// "ED"(Y error), "Z". If the data range is not belong to dependent
// or independent type, default can be "X".
dr.Add("X", wks, r1, c1, r2, c2);
```

Get data from data range to vector. **DataRange::GetData** supports multiple overloaded methods. For example:

```
vector vData;
int index = 0; // range index
dr.GetData(&vData, index);
```

Data Range in Matrixsheet

For a Matrix window, the data range can be a matrix object index.

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

DataRange dr;
int nMatrixObjectIndex = 0;
dr.Add(ml, nMatrixObjectIndex, "X");
```

Get data from data range to matrix.

```
matrix mat;
dr.GetData(mat);
```

Data Range in Graph

For a Graph window, the data range can be one data plot, or a sub range of one data plot.

```
GraphLayer gl = Project.ActiveLayer();
DataPlot dp = gl.DataPlots(); // Get active data plot

DataRange dr;
int i1 = 0; // from the first data point
int i2 = -1; // to the last data point
dp.GetDataRange(dr, i1, i2);
```

Get XY data from data plot to vector by data range object.

```
vector vx, vy;
DWORD dwRules = DRR_GET_DEPENDENT;
dr.GetData(dwRules, 0, NULL, NULL, &vy, &vx);
```

Data Range Control

OriginC supports a GetN dialog interactive control to choose a data range.

```
#include <GetNBox.h>
// Open a dialog to choose a range from one graph data plot.
// And construct a data range object by this selection.
GETN_TREE(tr)
GETN_INTERACTIVE(Range1, "Select Range", "")
if( GetNBox(tr) ) // returns true if click OK button
{

        DataRange dr;
        dr.Add("Range1", tr.Range1.strVal);

        vector vData;
        int index = 0; // range index
        dr.GetData(&vData, index); // The data in vData is the selected
data points
}
```

7.2 String Data

7.2.1 String Variables

```
string str1; // Declare a string variable named str1
str1 = "New York"; // Assigns to str1 a character sequence

string str2 = "Tokyo"; // Declare a string variable and assignment

// Declare a character array and initialize with a character sequence
char ch[] = "This is a test!";

// Declare a character array, set size and initialize with a character
sequence
char chArr[255] = "Big World.";
```

7.2.2 Convert String to Numeric

```
string str = PI; // Assigns a numeric value to string variable
```

```
// Convert string to numeric
double dd = atof(str, true);
out_double("dd=", dd); // dd=3.14159

// Convert string to complex
str = "1+2.5i";
complex cc = atoc(str);
out_complex("cc = ", cc); // cc = 1.000000+2.500000i

// Convert string to int
str = "100";
int nn = atoi(str);
out_int("nn = ", nn); // nn = 100
```

7.2.3 Append Numeric/String to another String

```
// Append numeric or string to another string
// In Origin C, support use '+' to add a numeric/string type const or
variable
string str = "The area is " + 30.7; // Append a double type const to
string

str += "\n"; // Append a string const to string variable

int nLength = 10;
str += "The length is " + nLength; // Append a int type variable to
string

out_str(str);
```

7.2.4 Find Sub String

```
// Find and get sub string
string str = "[Book1]Sheet1!A:C";
int begin = str.Find(']'); // Find and return the index of ']'
begin++; // Move to the next character of ]

int end = str.Find('!', begin); // Find and return the index of '!'
end--; // Move the previous character of !

// Get the sub string with the begin index and substring length
int nLength = end - begin + 1;
string strSheetName = str.Mid(begin, nLength);
out_str(strSheetName);// Should output "Sheet1"
```

7.2.5 Replace Sub String

```
// Find and replace one character
string str("A+B+C+");
int nCount = str.Replace('+','-');
out_int("", nCount); // nCount will be 3
out_str(str); // "A-B-C-"

// Find and replace a character string
str = "I am a student.\nI am a girl.";
nCount = str.Replace("I am", "You are");
```

```
out_int("", nCount); // nCount will be 2
out_str(str);
```

7.2.6 Path String Functions

File Path String

```
// string::IsFile is used to check the file if exist
string strFile = "D:\\TestFolder\\abc.txt";
bool bb = strFile.IsFile();
printf("The file %s is %sexist.\n", strFile, bb ? "" : "NOT ");

// GetFilePath function is used to extract the path from a full path
string
string strPath = GetFilePath(strFile);
out_str(strPath);

// GetFileName function is used to extracts the file name part
// from a string of full path
bool bRemoveExtension = true;
string strFileName = GetFileName(strFile, bRemoveExtension);
out_str(strFileName);

// string::IsPath to check if the path is exist
bb = strPath.IsPath();
out_int("", bb);
```

Origin System Path

```
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);
```

7.3 Date and Time Data

Origin C provides support for date and time data.

7.3.1 Get Current Date Time

```
// Get current time
time_t aclock;
time( &aclock );

// Converts a time value and corrects for the local time zone
TM tmLocal;
convert_time_to_local(&aclock , &tmLocal);

// Convert time value from TM format to system time format
SYSTEMTIME sysTime;
tm_to_systemtime(&tmLocal, &sysTime);
```

```
// Get date string from system time
char    lpcstrTime[100];
if(systemtime_to_date_str(&sysTime, lpcstrTime,
LDF_SHORT_AND_HHMM_SEPARCOLON))
    printf("Current Date Time is %s\n", lpcstrTime);
```

7.3.2 Convert Julian Date to String

```
SYSTEMTIME st;
GetSystemTime(&st); // Gets current date time

double dJulianDate;
SystemTimeToJulianDate(&dJulianDate, &st); // Convert to Julian date

// Convert Julian date to string with the specified format
string strDate = get_date_str(dJulianDate,
LDF_SHORT_AND_HHMM_SEPARCOLON);
out_str(strDate);
```

7.3.3 Convert String to Julian Date

```
string strDate = "090425 17:59:59";
double dt = str_to_date(strDate, LDF_YYMMDD_AND_HHMMSS);
```

8 Projects

The Origin C **Project** class is used for accessing the various high level objects contained in an Origin project. This includes workbooks, matrixbooks, graphs, notes, folders, and more.

8.1 Managing Projects

Origin C provides the Project class for opening, saving, and appending projects and for accessing the various objects contained in a project. The Project class contains collections for all the page types and loose data sets. There are methods to get the active objects such as the active curve, layer, and folder.

8.1.1 Open and Save a Project

The code below demonstrates saving a project, starting a new project, and opening a saved project.

```
string strPath = "c:\\abc.opj"; // Project path and name

Project.Save(strPath); // Save current project
Project.Open();        // Start a new project
Project.Open(strPath); // Open saved project
```

8.1.2 Append One Project to Another

You can append a project to the current project by using the optional second argument of the **Project::Open** method. The appended project's folder structure will be put into the current project's active folder.

```
Project.Open("c:\\abc.opj", OPJ_OPEN_APPEND);
```

8.1.3 The Modified Flag

When a project is modified, the **IsModified** flag is set internally by Origin. Origin C allows setting and clearing the **IsModified** flag. When a project is being closed, this flag is checked. If the flag is set then Origin will ask the user if they want to save their changes. If your Origin C code made changes that you know should not be saved, then you may want to clear the flag to prevent Origin from prompting the user.

```
if( Project.IsModified() )
{
    // Set the active project as not modified. We do this when we know
    // we do not want to save the changes and want to prevent Origin
    // from prompting the user about unsaved changes.
    Project.ClearModified();

    // Start a new project, knowing the user will not be prompted about
    // unsaved changes in the active project.
    Project.Open();
}
```

8.2 Managing Folders

Pages in an Origin project (workbooks, matrix books, and graphs) can be organized in a hierarchical folder structure, visible in Origin's Project Explorer. The Origin C Folder class allows you to create, activate, select, and arrange folders.

8.2.1 Create a Folder and Get Its Path

```
Folder fldRoot, fldSub;
fldRoot = Project.RootFolder;

// Add a sub folder in root folder with name
fldSub = fldRoot.AddSubfolder("MyFolder");
printf("Folder added successfully, path is %s\n", fldSub.GetPath());
```

8.2.2 Get the Active Folder

```
Folder fldActive;
fldActive = Project.ActiveFolder();

// Add a sub folder to it
fldSub = fldActive.AddSubfolder("MyFolder");
printf("Folder added successfully, path is %s\n", fldSub.GetPath());
```

8.2.3 Activate a Folder

```
// activate root folder
Folder fldRoot = Project.RootFolder;
fldRoot.Activate();

// activate the specified sub folder
Folder fldSub("/MyFolder");
fldSub.Activate();
```

8.2.4 Get Path for a Specific Page

```
GraphPage gp("Graph1");
if( gp.IsValid() )
{
        Folder fld = gp.GetFolder();
        out_str(fld.GetPath());
}
```

8.2.5 Move a Page/Folder to Another Location

Folder::Move is used to move a window (Worksheet, Graph...) or folder to another location. The following example shows how to move a folder.

```
// Add two sub folders to root folder
Folder subfld1 = root.AddSubfolder("sub1");
Folder subfld2 = root.AddSubfolder("sub2");

// Move the sub2 folder under the sub1 folder
if( !root.Move(subfld2.GetName(), "/"+subfld1.GetName()+"/", true) )
    printf("move folder failed!");
```

8.3 Accessing Pages

Pages in Origin consist of workbooks, matrix books and graphs, and are the core objects in a project. Origin C allows you to access a page by name or by index, or access all instances of a particular page type in the current project using the **foreach** statement.

8.3.1 Access a Page by Name and Index

All pages have names, which can be used to access them, as in the following example:

```
// Access a page by its name
GraphPage gp1("Graph1");

// Access a page by its zero based index
GraphPage gp2 = Project.GraphPages(0); // 0 for first page
```

8.3.2 Get the Active Page and Layer

In a workbook page, a layer is a worksheet; in a graph page, a layer is a pair of axes; in a matrix page, a layer is a matrix sheet.
If you want to access the page associated with a particular layer, such as the active layer, it can be done with the **Layer::GetPage** method:

```
// get active layer
GraphLayer gl = Project.ActiveLayer();

// get active page from layer
GraphPage gp = gl.GetPage();
```

8.3.3 Using foreach

The **foreach** statement simplifies the process of looping through all the items in a collection. The project contains all the pages in various collections.

```
// Loop through all workbook pages in the current project
// and output the name of each page.
foreach( WorksheetPage wksPage in Project.WorksheetPages )
{
    out_str(wksPage.GetName());
}
// Loop through all matrixbook pages in the current project
// and output the name of each page.
foreach( MatrixPage matPage in Project.MatrixPages )
```

```
{
    out_str(matPage.GetName());
}
// Loop through all graph pages in the current project
// and output the name of each page.
foreach( GraphPage gp in Project.GraphPages )
{
    out_str(gp.GetName());
}
// Loop through all pages in the current project
// and output the name of each page.
foreach( Page pg in Project.Pages )
{
    out_str(pg.GetName());
}
```

8.4 Accessing Metadata

Metadata is information which refers to other data. Examples include the time at which data was originally collected, the operator of the instrument collecting the data and the temperature of a sample being investigated. Metadata can be stored in Projects, Pages, Layers and Columns.

8.4.1 Access DataRange

The Origin C Project class provides methods to add, get, and remove an Origin C DataRange object to and from the current project.

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

DataRange dr;                      // Construct the range object
dr.Add("X", wks, 0, 0, -1, -1); // Add whole worksheet to range
dr.SetName("Range1");              // Set range name
int UID = dr.GetUID(TRUE);         // Get Unique ID for the range object

int nn = Project.AddDataRange(dr); // Add range to project
```

In the Command Window or Script Window you can use the LabTalk command **list r** to list all the DataRange objects in the current project.

8.4.2 Access Tree

Access a Tree in a Project

Add Tree

This code declares a variable of type tree, assigns some data to nodes of the tree, and adds the tree to the current project.

```
Tree tr;
```

```
tr.FileInfo.name.strVal = "Test.XML";
tr.FileInfo.size.nVal = 255;

// add tree variable to project
int nNumTrees = Project.AddTree("Test", tr);
out_int("The number of trees in project: ", nNumTrees);
```

Get Tree

Likewise, a similar code extracts data stored in an existing tree variable named *Test* and puts it into a new tree variable named *trTest*:

```
// get tree from project by name
Tree trTest;
if( Project.GetTree("Test", trTest) )
        out_tree(trTest);
```

Get the Names of All LabTalk Trees

The **Project::GetTreeNames** method gets the names of all LabTalk tree variables in the project. Here, the names are assigned to a string vector; the number of strings assigned is returned as an integer.

```
vector<string> vsTreeNames;
int nNumTrees = Project.GetTreeNames(vsTreeNames);
```

Access Tree in a Worksheet

OriginObject::PutBinaryStorage is used to put a tree into many types of Origin object, for example, a WorksheetPage, Worksheet, Column, GraphPage, or MatrixPage.

Add Tree

Keep an active worksheet window in the current project, to run the example code below. After running the code to add a user tree, right click on the title of the worksheet window, choose Show Organizer, and you will see the added user tree show up in the panel on the right.

```
Worksheet wks = Project.ActiveLayer();
if( wks )
{
        Tree tr;
        tr.name.strVal = "Jacky";
        tr.id.nVal = 7856;

        // put tree with name wksTree to worksheet object
        string strStorageName = "wksTree";
        wks.PutBinaryStorage(strStorageName, tr);
}
```

Get Tree

The **OriginObject::GetBinaryStorage** method is used to get a tree from an Origin object by name.

```
Worksheet wks = Project.ActiveLayer();
if( wks )
{
        Tree tr;
        string strStorageName = "wksTree";

        // if the tree named wksTree is existed, return true.
        if( wks.GetBinaryStorage(strStorageName, tr) )
                out_tree(tr); // output tree
}
```

Get the Names of All Trees

The **OriginObject::GetStorageNames** method gets the names of everything in storage in an Origin object. There are two storage types: INI and binary. Trees belong to binary storage, and the following example code shows how to get binary storage from a Worksheet.

```
Worksheet wks = Project.ActiveLayer();
if( wks )
{
        // get the names of all binary type storage
        vector<string> vsNames;
        wks.GetStorageNames(vsNames, STORAGE_TYPE_BINARY);

        for(int nn = 0; nn < vsNames.GetSize(); nn++)
                out_str(vsNames[nn]);
}
```

Access Tree in a Worksheet Column

For setting and getting a tree in a Worksheet Column, use the same methods for setting and getting a tree in a Worksheet, as described above.

Add Tree

```
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);
```

Get Tree

```
Worksheet wks = Project.ActiveLayer();
Column col(wks, 0);

Tree tr;
if( col.GetBinaryStorage("colTree", tr) )
```

```
        out_tree(tr);
```

Get the Names of All Trees

```
Worksheet wks = Project.ActiveLayer();
Column col(wks, 0);

// get the names of all binary type storage
vector<string> vsNames;
col.GetStorageNames(vsNames, STORAGE_TYPE_BINARY);

for(int nn = 0; nn < vsNames.GetSize(); nn++)
        out_str(vsNames[nn]);
```

Access Import File Tree Nodes

After importing data into a worksheet, Origin stores metadata in a special tree-like structure at the page level. Basic information about the file can be retrieved and put into a tree.

```
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);
```

Access Report Sheet Tree

Analysis Report sheets are specially formatted Worksheets based on a tree structure. You can get the report tree from a report sheet as below.

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

Tree trReport;
uint uid; // to receive the UID of the report range
// true to translate the escaped operation strings(ex. ?$OP:A=1)
// to real dataset name in the returned tree
bool bTranslate = true;
if( wks.GetReportTree(trReport, &uid, 0, GRT_TYPE_RESULTS, true) )
{
        out_tree(trReport);
}
```

8.5 Accessing Operations

8.5.1 List All Operations

Many recalculating analysis tools, such as the Statistics on Columns dialog, the Nonlinear Curve Fitting dialog, etc., are based on the Operation class. After finishing the whole operation, there will be a lock on the result sheet or result graph. We can list all operations via **Project::Operations**. The following code is used to get all operations objects and print out the operation names.

```
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);
}
```

8.5.2 Check Worksheet if Hierarchy

If you want to check whether a worksheet is a result table sheet, you can check with layer system parameters, as in the following code.

```
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");
```

8.5.3 Accessing Report Sheet

The following code shows how to get a report tree from a report sheet, convert the result gotten from the report tree into a cell linking format string, and put it into a new worksheet.
This is how to get a report tree from a report sheet. To run this code you need keep a report sheet active.

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

Tree trResult;
wks.GetReportTree(trResult);
```

The following code shows how to get the needed results from the report tree, convert them to a cell linking format string, and put it into a newly created worksheet.

```
// Add a new sheet for summary table
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;
// Parameters
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");

// Statistics
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");

// put to columns
Column colLabel(wksSummary, 0);
Column colValue(wksSummary, 1);
colLabel.PutStringArray(vsLabels);
colValue.PutStringArray(vsValues);
```

9 Importing

One of the huge benefits of Origin is the ability to import data of different formats into a worksheet or a matrix sheet. Origin C provides this ability to import ASCII and binary data files, image files, and data from a database. The following sections will show you how to import data into a worksheet or matrix sheet.

9.1 Importing Data

The Worksheet and MatrixLayer classes are derived from the Datasheet class. The Datasheet class has a method named ImportASCII. The ImportASCII method is used for importing ASCII data files. There are also ImportExcel and ImportSPC methods for importing Microsoft Excel and spectroscopic data files, respectively.

9.1.1 Import ASCII Data File into Worksheet

The first example will import an ASCII data file into the active worksheet of the active workbook. It will first call the AscImpReadFileStruct global function to detect the file's format. The format information is stored in an ASCIMP structure. The structure will then be passed to the ImportASCII method to do the actual importing.

```
string strFile = "D:\\data.dat"; // some data file name
ASCIMP  ai;
if(0 == AscImpReadFileStruct(strFile, &ai) )
{
    // In this example we will disable the ASCII import progress
    // bar by setting the LabTalk System Variable @NPO to zero.
    // This is optional and is done here to show it is possible.
    // The LTVarTempChange class makes setting and restoring a
    // LabTalk variable easy.  See the Accessing LabTalk section
    // for more details about the LTVarTempChange class.
    LTVarTempChange progressBar("@NPO", 0); // 0 = disable progress bar

    // Get active worksheet from active work book.
    Worksheet wks = Project.ActiveLayer();

    if(0 == wks.ImportASCII(strFile, ai))
        out_str("Import data successful.");
}
```

The next example will also import an ASCII data file into a worksheet but it will also obtain additional information about each column from the file, and set up the worksheet columns.

```
// Prompt user with a File Open dialog to choose a file to import.
string  strFile = GetOpenBox("*.dat");
if( strFile.IsEmpty() )
    return; // User canceled or error

ASCIMP  ai;
```

```
if( 0 == AscImpReadFileStruct(strFile, &ai) )
{
    ai.iAutoSubHeaderLines = 0; // Disable auto detect sub header

    // 1, LongName
    // 2. Units
    // 3. Expanded Description(User defined)
    // 4. Type Indication(User defined)
    ai.iSubHeaderLines = 4;

    // When iAutoSubHeaderLines is false(0), the beginning index of
ai.nLongName,
    // ai.nUnits and ai.nFirstUserParams are from main header
    ai.nLongNames = ai.iHeaderLines;
    ai.nUnits = ai.iHeaderLines + 1;

    // Set the index for the first user params
    ai.nFirstUserParams = ai.iHeaderLines + 2;
    ai.nNumUserParams = 2; // Set the number of user params

    // Not set any header to Comments label
    ai.iMaxLabels = 0;

    // Get active worksheet from active work book.
    Worksheet wks = Project.ActiveLayer();

    if( 0 == wks.ImportASCII(strFile, ai) ) // Return 0 for no error
    {
        // The names of the user parameter labels
        vector<string> vsUserLabels = {"Expanded Description", "Type
Indication"};

        // Set user parameter labels to specified names
        Grid grid;
        grid.Attach(wks);
        grid.SetUserDefinedLabelNames(vsUserLabels);

        wks.AutoSize(); // Resize column widths to best fit their
contents.
    }
}
```

9.1.2 Import ASCII Data File into Matrix Sheet

Importing data into a matrix sheet is very similar to importing into a worksheet. This example is almost identical to the first worksheet example. The only difference is we get the active matrix sheet from the active matrix book using the MatrixLayer class instead of the Worksheet class.

```
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.");
}
```

9.1.3 Import Data Using an Import Filter

Functions for importing files are declared in the **OriginC\Originlab\FileImport.h** file. These functions are also documented in the Origin C Language Reference help file.
Prior to calling the import file functions, you need to first programmatically load and compile **FileImport.c**. This can be done from script using the command:

```
run.LoadOC(Originlab\FileImport.c, 16);
// Option 16 ensures that all dependent Origin C files are loaded,
// by scanning for the corresponding .h in FileImport.c
```

The following example shows importing data with a filter file.

```
#include <..\Originlab\FileImport.h>
void import_with_filter_file()
{
    Page pg = Project.Pages(); // Active Page

    // Get page book name
    string strPageName = pg.GetName();

    // Get page active layer index
    int nIndexLayer = pg.Layers().GetIndex();

    // Get Origin sample folder
    string strPath = GetAppPath(TRUE) + "Samples\\Signal Processing\\";

    // specify .oif filter name
    string strFilterName = "TR Data Files";

    import_file(strPageName, nIndexLayer, strPath + "TR2MM.dat",
strFilterName);
}
```

Sometimes the existing filter might need to be modified to meet the requirements of the data format, so you need to load the filter from the file and configure it. See the following case:

```
#include <..\Originlab\FileImport.h>
void    config_filter_tree()
{
    string strFile = GetAppPath(1) + "Samples\\Curve
Fitting\\Step01.dat";
    if( !strFile.IsFile() )
        return;

    // load filter to tree
    Tree trFilter;
    string strFilterName = "ASCII";
    int nLocation = 1; // build-in Filters folder
    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");

    // update filter tree
    trFilter.iRenameCols.nVal = 0; // 0 to keep default column name, 1
to rename column

    // import data file with filter tree.
    // import_files function supports import multiple files one time.
    vector<string> vsDataFileName;
    vsDataFileName.Add(strFile);
    nRet = import_files(vsDataFileName, strPageName, wks.GetIndex(),
trFilter);
    if( 0 != nRet )
        out_str("Failed to import file");
}
```

There are times when there is no existing filter for importing a data file. In these cases you will need to make an import filter using Origin's Import Wizard. Using Origin C and the impFile X-Function you can import data files using both Origin's included import filters and your own.
For a detailed example of calling the impFile X-Function from Origin C, please refer to the Calling X-Functions in Origin C section.

9.2 Importing Images

Origin allows you to import images into a matrix or a worksheet cell, and onto a graph. The following sections will show you how to import images in your Origin C applications.

9.2.1 Import Image into Matrix

The following example function demonstrates how to import an image file into a matrix. The function takes three arguments: matrix name, file name, and grayscale depth. The key functions being called in this example are oimg_image_info and oimg_load_image. The first is used to get information about the image contained in the image file. The information obtained is used in preparing the target matrix. The latter function is used to do the actual importing of the image file into the target matrix as grayscale data values.

```
#include <import_image.h> // needed for oimg_ functions

bool import_image_to_matrix_data(
    LPCSTR lpcszMatrixName, // matrix book name
    LPCSTR lpcszFileName,   // image file name
    int nGrayDepth)         // import as 8-bit or 16-bit gray
{
    // Get the target matrix object
    MatrixObject mo(lpcszMatrixName);
    if( !mo.IsValid() )
        return false;
    // Get source image information
    int nWidth, nHeight, nBPP;
    if( !oimg_image_info(lpcszFileName, &nWidth, &nHeight, &nBPP) )
        return false;

    // Set target matrix to same dimensions as source image
    if( !mo.SetSize(nHeight, nWidth, 0) )
        return false;

    // Set target matrix data size
    int nDataType = (16 == nGrayDepth ? FSI_USHORT : FSI_BYTE);
    if( !mo.SetInternalData(nDataType, FALSE, FALSE) )
        return false;

    // Import the image into the matrix
    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;
}
```

9.2.2 Import Image into Worksheet Cell

The following example will embed a JPEG image from a file into a worksheet cell. This is accomplished using the AttachPicture method of the Worksheet class.

```
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();
}
```

9.2.3 Import Image to Graph

The following example will embed a JPEG image from a file onto a graph layer. This is accomplished using the **image_import_to_active_graph_layer** global function.

```
#include <image_utils.h>

// make sure image_utils.c is compiled before calling
// the image_import_to_active_graph_layer function.
LT_execute("run.LoadOC(Originlab\\image_utils.c)");
```

```
string strFile = "D:\\Graph1.jpg";
image_import_to_active_graph_layer(strFile);
```

10 Exporting

10.1 Exporting Worksheets

The **Worksheet** class has the **ExportASCII** method for saving worksheet data to a file. The method has arguments for specifying the starting row and column and the ending row and column. It also allows you to specify how to handle missing data values and whether column labels should be exported or not.
All of the examples below assume wks is a valid **Worksheet** object and strFileName is a **string** object containing the full path and name of the target file.
The first example will save all the data in the worksheet to a file using the tab character as the delimiter and blanks for missing values.

```
wks.ExportASCII(strFileName,
    WKS_EXPORT_ALL|WKS_EXPORT_MISSING_AS_BLANK);
```

The next example will save all the data in a worksheet to a file, with a comma as the delimiter and blanks for missing values. In addition the column labels are also saved.

```
wks.ExportASCII(strFileName,
    WKS_EXPORT_ALL|WKS_EXPORT_LABELS|WKS_EXPORT_MISSING_AS_BLANK,
    ',');
```

The final example will save the first two columns of data in a worksheet to a file, using a comma as the delimiter and blanks for missing values. In addition, the column labels are also saved. Row and column indices start with zero. The end row and column indices can also be -1 to indicate the last row or last column, respectively.

```
wks.ExportASCII(strFileName,
    WKS_EXPORT_ALL|WKS_EXPORT_LABELS|WKS_EXPORT_MISSING_AS_BLANK,
    '\t',
    0, 0,   // start with first row, first column
    -1, 1); // end with last row, second column
```

10.2 Exporting Graphs

Origin allows users to export graphs to several different image file types. Origin C allows access to this ability using the global **export_page** and **export_page_to_image** functions.
The following example will export all the graphs in the project to EMF files. The EMF file names will be the same as the graph names, and will be located in the root of drive C.

```
string strFileName;
foreach(GraphPage gp in Project.GraphPages)
{
    strFileName.Format("c:\\%s.emf", gp.GetName());
    export_page(gp, strFileName, "EMF");
}
```

The next example will export the active graph to an 800x600 JPEG file. The JPEG file name will be the name of the graph and will be located in the root of drive C.

```
GraphPage gp;
gp = Project.ActiveLayer().GetPage();
if( gp ) // if active page is a graph
{
    string strFileName;
    strFileName.Format("c:\\%s.emf", gp.GetName());
    export_page_to_image(strFileName, "JPG", gp, 800, 600);
}
```

10.3 Exporting Matrices

An Origin Matrix can be exported to an ASCII data file or an image file.

10.3.1Export Matrix to ASCII Data File

The following example shows how to export ASCII data from the active matrix window to a *.txt file. You need to include the oExtFile.h file for the **export_matrix_ascii_data** function.

```
file     ff;
if ( !ff.Open("C:\\ExpMatData.txt", file::modeCreate|file::modeWrite) )
    return; //fail to open file for write

string         strRange;
MatrixLayer ml = Project.ActiveLayer();
ml.GetRangeString(strRange);

LPCSTR                 lpcszSep = "\t";
vector<string> vXLabels, vYLabels; // empty means no label
DWORD                  dwCntrl = GDAT_FULL_PRECISION |
GDAT_MISSING_AS_DASHDASH;

// return 0 for no error
int nErr = export_matrix_ascii_data(&ff, strRange, ml.GetNumRows(),
        ml.GetNumCols(), lpcszSep, &vXLabels, &vYLabels, dwCntrl);
```

10.3.2Export Image from Matrix to Image File

The following example shows how to export a matrix to an image file. You need to include the image_utils.h file for the **export_Matrix_to_image** function.

```
MatrixLayer ml = Project.ActiveLayer();
MatrixObject mo = ml.MatrixObjects();
export_Matrix_to_image("c:\\matrixImg.jpg", "jpg", mo);
```

11 Analysis and Applications

Origin C supports functions that are valuable to data analysis, as well as mathematic and scientific applications. The following sections provide examples on how to use the more common of these functions, broken down by categories of use.

11.1 Mathematics

11.1.1 Normalize

The following example shows how to pick a point in a data plot (curve) and then normalize all curves in the layer to the same value at that point. This snippet of code assumes a graph layer with multiple curves is active and all curves share the same X values. This assumption is typical in spectroscopy.

```
GraphLayer gl = Project.ActiveLayer();
if( !gl )
    return;

// Allow user to click and select one particular point of one
particular curve
GetGraphPoints mypts;
mypts.SetFollowData(true);
mypts.GetPoints(1, gl);

vector vx, vy;
vector<int> vn;
if(mypts.GetData(vx, vy, vn) == 1)
{
    // Save index and y value of picked point
    int nxpicked = vn[0] - 1;
    double dypicked = vy[0];

    // Loop over all data plots in layer
    foreach( DataPlot dp in gl.DataPlots )
    {
        // Get the data range and then the y column for current plot
        XYRange xy;
        Column cy;
        if(dp.GetDataRange(xy) && xy.GetYColumn(cy))
        {
            // Get a vector reference to y values from the y column
            vectorbase &vycurrent = cy.GetDataObject();

            // Scale vector so y value matches user-picked point
            vycurrent *= dypicked/vycurrent[nxpicked];
        }
    }
}
```

11.1.2 Interpolation / Extrapolation

The **ocmath_interpolate** function is used to do interpolation/extrapolation with modes of Linear, Spline and B-Spline.

```
// Make sure there are 4 columns in active worksheet
// The first two columns are source xy data,
// 3rd column has input x data, 4th column to put output y.
Worksheet    wks = Project.ActiveLayer();
wks.SetSize(-1, 4);

DataRange drSource;
drSource.Add(wks, 0, "X"); // 1st column - source x data
drSource.Add(wks, 1, "Y"); // 2nd column - source y data

vector vSrcx, vSrcy;
drSource.GetData(&vSrcx, 0);
drSource.GetData(&vSrcy, 1);

DataRange drOut;
drOut.Add(wks, 2, "X"); // 3rd column - input x data
drOut.Add(wks, 3, "Y"); // 4th column - interpolated y data

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);
```

11.1.3 Integration

Origin C provides access to NAG's integral routines to perform integration. With Origin C and NAG you can do integration on a normal integrand, an integrand with parameters, an integrand with oscillation, an infinite integral, higher dimension integration, and more. The following examples show how to do integration with NAG. Your Origin C code will need to include the NAG header file at least once before your code calls any NAG functions.

```
#include <OC_nag8.h> // NAG declarations
```

Simple Integral Function

The first example shows how to do a basic integration on a simple integrand with only one integration variable.

```
// NAG_CALL denotes proper calling convention. You may treat it
// like a function pointer and define your own integrand
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;  // integration interval

        double epsabs, abserr, epsrel, result;
        // you may use epsabs and epsrel and this quantity to enhance
your desired
        // precision when not enough precision encountered
        epsabs = 0.0;
        epsrel = 0.0001;

        // The max number of sub-intervals needed to evaluate the
function in the
        // integral. For most cases 200 to 500 is adequate and
recommmended.
        int max_num_subint = 200;

        Nag_QuadProgress qp;
        NagError fail;

        d01ajc(func, a, b, epsabs, epsrel, max_num_subint, &result,
&abserr,
        &qp, &fail);


        // For the error other than the following three errors which are
due to
        // bad input parameters or allocation failure. You will need to
free
        // the memory allocation before calling the integration routine
again
        // to avoid memory leakage
        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);
}
```

Integral Function with Parameters

The next example shows how to define and perform integration on an integrand with parameters. Notice that the parameters are passed to the integrator by a user-defined structure. This avoids having to use static variables as parameters of the integrand, and makes it thread-safe.
This example can also be adapted to use NAG's infinite integrator. For instance, by enabling the line calling the infinite integrator **d01smc** function, the example can be used to perform infinite integration.

```
struct user // integrand parameters
{
    double A;
    double Xc;
    double W;
};

// Function supplied by user, return the value of the integrand at a
given x.
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));
}
```

Now, we set parameter values for the function and define the additional parameters necessary to perform the integration. The integration is then performed by a single function call, passing the parameters as arguments.

```
void nag_d01sjc_ex()
{
    double a = 0.0;
    double b = 2.0; // integration interval

    // The following variables are used to control
    // the accuracy and precision of the integration.
    double epsabs = 0.0;      // absolute accuracy, set negative to use
relative
    double epsrel = 0.0001;   // relative accuracy, set negative to use
absolute
    int max_num_subint = 200; // max sub-intervals, 200 to 500 is
recommended

    // Result keeps the approximate integral value returned by the
algorithm
    // abserr is an estimate of the error which should be an upper
bound
    // for |I - result| where I is the integral value
    double result, abserr;

    // The structure of type Nag_QuadProgress, it contains pointers
    // allocated memory internally with max_num_subint elements
    Nag_QuadProgress qp;

    // The NAG error parameter (structure)
    NagError fail;

    // Parameters passed to integrand by NAG user communication struct
    struct user param;
    param.A  = 1.0;
    param.Xc = 0.0;
    param.W  = 1.0;

    Nag_User comm;
    comm.p = (Pointer)&param;

    // Perform integration
    // There are 3 kinds of infinite boundary types you can use in Nag
infinite
    // 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);

    // check the error by printing out error message
    if (fail.code != NE_NOERROR)
        printf("%s\n", fail.message);

    // For errors other than the following three errors which are due
to
    // bad input parameters, or allocation failure,
    // you will need to free the memory allocation before calling the
    // integration routine again to avoid memory leakage.
    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);
}
```

Multi-dimension Integral Function

For integrals of dimension higher than 2, you can call the NAG integrator function **d01wcc** to perform the integration.
Our user defined call back function will be passed to the NAG **d01wcc** function.

```
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);
}
```

Main function:

```
void nag_d01wcc_ex()
{
        // Input variables
        int ndim = NDIM;  // the integral dimension
        double a[4], b[4];
        for(int ii=0; ii < 4; ++ii)  // integration interval
        {
                a[ii] = 0.0;
                b[ii] = 1.0;
        }
        int minpts = 0;
        int maxpts = MAXPTS;  // maximum number of function evaluation
        double eps = 0.0001; // set the precision

        // Output variable
        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);
        }
}
```

11.1.4 Differentiation

The **ocmath_derivative** function is used to do simple derivative calculations without smoothing. The function is declared in ocmath.h as shown below.

```
int ocmath_derivative(
    const double* pXData, double* pYData, uint nSize, DWORD dwCntrl =
0);
```

The function ignores all missing values and computes the derivative by taking the average of the two slopes between the point and each of its neighboring data points. If the dwCntrl argument uses the default value of 0, the function fills in the average when data changes direction.

```
if( OE_NOERROR == ocmath_derivative(vx, vy, vx.GetSize()) )
    out_str("successfully");
```

If dwCntrl is set to DERV_PEAK_AS_ZERO, the function fills in zero if data changes direction.

```
if( OE_NOERROR == ocmath_derivative(vx, vy, vx.GetSize(),
DERV_PEAK_AS_ZERO) )
    out_str("successfully");
```

11.2 Statistics

Often we want to do statistics on the selected data in a worksheet, i.e. one column, one row, or an entire worksheet. The **Working with Data: Numeric Data: DataRange** chapter shows how to construct a data range object by column/row index, then get the raw data into a vector.

11.2.1 Descriptive Statistics on Columns and Rows

The **ocmath_basic_summary_stats** function is used to compute basic descriptive statistics, such as total number, mean, standard deviation, and skewness, for raw data. For more details, refer to Origin C help. The following Origin C code calculates and outputs the number of points, the mean, and the standard error of mean on the data in the vector object named vData.

```
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);
```

11.2.2 Frequency Count

The **ocmath_frequency_count** function is used to calculate the frequency count, according to the options in the FreqCountOptions structure.

```
// Source data to do frequency count
vector vData = {0.11, 0.39, 0.43, 0.54, 0.68, 0.71, 0.86};

// Set options, including bin size, from, to and border settings.
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; // to extend last bin if not a full bin

int nRet = ocmath_frequency_count(
    vData, vData.GetSize(), &fcoOptions,
    vBinCenters, nBinSize, vAbsoluteCounts, nBinSize,
    vCumulativeCounts, nBinSize, nOption);

if( STATS_NO_ERROR == nRet )
    out_str("Done");
```

In addition, there are two functions to calculate frequency count for discrete/categorical data. One is ocu_discrete_frequencies for text data, and the other is ocmath_discrete_frequencies for numeric data. Also, there are two functions to calculate frequency count on 2 dimensions: ocmath_2d_binning_stats and ocmath_2d_binning.

11.2.3Correlation Coefficient

The **ocmath_corr_coeff** function is used to calculate the Pearson rank, Spearman rank and Kendall rank correlation coefficients.

```
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");
}
```

11.2.4Normality Test

Use the ***ocmath_shapiro_wilk_test** function to perform a Shapiro-Wilk Normality Test. Use the ***ocmath_lilliefors_test** function to perform a Lilliefors Normality Test. Use the ***ocmath_kolmogorov_smirnov_test** function to perform a Kolmogorov-Smirnov Normality Test.

```
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);
}
```

11.3 Curve Fitting

11.3.1Linear Fitting

To perform a linear fitting routine in Origin C, you can use the **ocmath_linear_fit** function. With this function, you can do linear fitting with weight, and then you can get the fitting results, including parameter values, statistical information, etc.
The following procedure will show how to perform linear fitting in Origin C by using this function, and the results will output to the specified windows and worksheet.

Perform Linear Fitting

Before starting linear fitting, please import the desired data, here need one independent and one dependent.
Now, begin the Origin C routine. Three steps are needed.

1. New a c file and add an empty function as the below. Copy the codes from the following steps into this function.

```
#include <GetNBox.h> // used for GETN_ macros

void linearfit()

{

}
```

2. Get the data from the worksheet for linear fit. Both independent and dependent are using vector variables.

```
// Get XY data from worksheet window

Worksheet wks = Project.ActiveLayer();

if(!wks)

    return;  // need to activate a worksheet with data

WorksheetPage wp = wks.GetPage();


DataRange dr;

dr.Add("X", wks, 0, 0, -1, 0);  // x column

dr.Add("Y", wks, 0, 1, -1, 1);  // y column


vector vX;

dr.GetData(&vX, 0);  // get data of x column to vector


vector vY;

dr.GetData(&vY, 1);  // get data of y column to vector
```

3. Show GetN dialog to control fit options and call ocmath_linear_fit function to do linear fit with these options.

```
// Prepare GUI tree to show fit options in GetN dialog

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; // clicked Cancel button

}

LROptions stLROptions;

stLROptions = trGUI.Fit; // assign value from GUI tree to struct



// Do linear fit with the above input dataset and fit option
settings
```

```
int nSize = vX.GetSize();  // data size

FitParameter psFitParameter[2];  // two parameters

RegStats stRegStats;  // regression statistics

RegANOVA stRegANOVA;  // anova statistics



int nRet = ocmath_linear_fit(vX, vY, nSize, psFitParameter, NULL,

                        0, &stLROptions, &stRegStats,
&stRegANOVA);

if(nRet != STATS_NO_ERROR)

{

    out_str("Error");

    return;

}
```

Result to Output Window

Once the computation is finished, the fitting results can be output to the specified windows. Here the values of parameters will output to the Script Window and the statistical information will output to the Result Log window as a tree.

```
void put_to_output_window(const FitParameter* psFitParameter,
                       const RegStats& stRegStats, const RegANOVA&
stRegANOVA)
{
        // Output analysis result to Script window, Result Log and
Worksheet
        // print the values of fitting parameters to the Script Window
        vector<string> vsParams = {"Intercept", "Slope"};
        for(int iPara = 0; iPara < vsParams.GetSize(); iPara++)
        {
            printf("%s = %g\n", vsParams[iPara],
psFitParameter[iPara].Value);
        }

        // Put the statistical results to Result Log
        Tree trResults;
        TreeNode trResult = trResults.AddNode("LinearFit");
        TreeNode trStats = trResult.AddNode("Stats");
        trStats += stRegStats;  // add regression statistics to tree
node

        TreeNode trANOVA = trResult.AddNode("ANOVA");
        trANOVA += stRegANOVA;  // add anova statistics to tree node
        string strResult;
        tree_to_str(trResult, strResult);  // convert tree to string

        Project.OutStringToResultsLog(strResult);  // output to Result
Log
}
```

Result to Worksheet

You can output the fitting result to the specified Worksheet as well. And the results can be organized in normal column format or tree view format in Worksheet window. The following two ways both used **Datasheet::SetReportTree** method to put result in Worksheet by tree variable. The difference is the option bit WP_SHEET_HIERARCHY when create worksheet, see the 2nd variable used **AddLayer** method below.

Output to Normal Worksheet

```
void output_to_wks(WorksheetPage wp, const FitParameter*
psFitParameter)
{
        // prepare report tree
        int nID = 100; // Each node must have node ID and node ID must
be unique
        Tree tr;
        tr.Report.ID = nID++;
        TreeNode trReport = tr.Report;
        trReport.SetAttribute(TREE_Table, GETNBRANCH_TRANSPOSE);

        // column 1
        trReport.P1.ID = nID++;
        trReport.P1.SetAttribute(STR_LABEL_ATTRIB, "Parameter"); //
column label
        trReport.P1.SetAttribute(STR_COL_DESIGNATION_ATTRIB,
OKDATAOBJ_DESIGNATION_X);

        // column 2
        trReport.P2.ID = nID++;
        trReport.P2.SetAttribute(STR_LABEL_ATTRIB, "Value"); // column
label
        trReport.P2.SetAttribute(STR_COL_DESIGNATION_ATTRIB,
OKDATAOBJ_DESIGNATION_Y);

        // column 3
        trReport.P3.ID = nID++;
        trReport.P3.SetAttribute(STR_LABEL_ATTRIB, "Prob>|t|"); //
column label
        trReport.P3.SetAttribute(STR_COL_DESIGNATION_ATTRIB,
OKDATAOBJ_DESIGNATION_Y);

        // prepare the vectors to show in the table
        vector<string> vsParamNames = {"Intercept", "Slope"};  //
parameter name
        vector vValues, vProbs;  // parameter value and prob
        for(int nParam = 0; nParam < vsParamNames.GetSize(); nParam++)
        {
            vValues.Add(psFitParameter[nParam].Value);
            vProbs.Add(psFitParameter[nParam].Prob);
        }

        // assign the vectors to tree node
        trReport.P1.strVals = vsParamNames;
        trReport.P2.dVals = vValues;
        trReport.P3.dVals = vProbs;

        // report tree to worksheet
        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();
}
```

Output to Tree Format Worksheet

```
void output_to_tree_view_wks(WorksheetPage& wp, const RegStats&
stRegStats)
{
        Tree tr;
        int nID = 100; // Each node must have node ID and node ID must
be unique
        uint nTableFormat = GETNBRANCH_OPEN
                                                        |
GETNBRANCH_HIDE_COL_HEADINGS
                                                        |
GETNBRANCH_HIDE_ROW_HEADINGS
                                                        | GETNBRANCH_FIT_COL_WIDTH
                                                        |
GETNBRANCH_FIT_ROW_HEIGHT;

        // prepare root table node
        tr.Report.ID = nID++; // add Report treenode and assign node id
        TreeNode trReport = tr.Report;
        // need set table attribute for table node
        trReport.SetAttribute(TREE_Table, nTableFormat);
        // the title of root table
        trReport.SetAttribute(STR_LABEL_ATTRIB, "Linear Fit Stats");

        // prepare stats table node
        trReport.Table.ID = nID++; // add Table treenode and assign node
id
        TreeNode trTable = trReport.Table;
        // need set table attribute for table node
        trTable.SetAttribute(TREE_Table,
nTableFormat|GETNBRANCH_TRANSPOSE);
        // the title of stats table
        trTable.SetAttribute(STR_LABEL_ATTRIB, "Regression Statistics");
        // prepare result node
        trTable.Stats.ID = nID++; // add Stats treenode and assign node
id
        TreeNode trStats = trTable.Stats;
        trStats += stRegStats; // support adding result from sturct to
treenode

        // set label, those text will show in row header in table
        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");

        // to hide other nodes
        trStats.ReducedChiSq.Show = false;
        trStats.Correlation.Show = false;
        trStats.Rvalue.Show = false;
        trStats.RSqCOD.Show = false;
        trStats.RMSESD.Show = false;
        trStats.NormResiduals.Show = false;

        // the bits to control the newly created worksheet as hierarchy
format
        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();
}
```

11.3.2Polynomial Fitting

To perform a polynomial fitting routine in Origin C, you can use the **ocmath_polynomial_fit** function. With this function, you can do polynomial fitting with weight, and then you can get the fitting results, including parameter values, statistical information, etc.
The following procedure will show how to perform polynomial fitting in Origin C by using this function.

Perform Polynomial Fitting

Before doing polynomial fitting, please import the desired data, here need one independent and one dependent.
The procedure of performing polynomial fitting needs three steps.

1. Get the data from the worksheet for polynomial fit. Both independent and dependent are using vector variables.

```
Worksheet wks = Project.ActiveLayer();
if(!wks)
	return;  // invalid worksheet

DataRange dr;
dr.Add("X", wks, 0, 0, -1, 0);  // x column
dr.Add("Y", wks, 0, 1, -1, 1);  // y column

vector vX, vY;
dr.GetData(&vX, 0);  // get data of x column to vector
dr.GetData(&vY, 1);  // get data of y column to vector
```

2. Define the structure variables and other data types as parameters for passing to the function. It also can initialize some fitting settings.

```
// here just define the structure for output results
int nSize = vX.GetSize();
const int nOrder = 2;  // order

int nSizeFitParams = nOrder+1;
FitParameter psFitParameter[3];  // number of parameter = nOrder+1

RegStats psRegStats;  // regression statistics
RegANOVA psRegANOVA;  // anova statistics
```

3. Pass the desired arguments and perform polynomial fitting on the data.

```
// polynomial fitting, using the default options, 2 order
int nRet = ocmath_polynomial_fit(nSize, vX, vY, NULL, nOrder, NULL,
psFitParameter,

	nSizeFitParams, &psRegStats, &psRegANOVA);

// check error
if(nRet!=STATS_NO_ERROR)
{
	out_str("Error");
	return;
}
```

Output the Results

After finishing the calculation, the results may need to output to somewhere for presentation, such as Script Window, Result Log, Worksheet, etc.
Please refer to the **Result to Output Window** and **Result to Worksheet** section in the chapter **Analysis and Applications: Curve Fitting: Linear Fitting** for more details about how to output the results.

11.3.3 Multiple Regression

Origin uses the **ocmath_multiple_linear_regression** function to perform multiple linear regression. This function allows to specify the weight of data and linear

regression options. After running this function successfully, output details will include fitting parameters, regression statistics, ANOVA statistics, covariance and correlation matrix of estimate, and etc.
In the following sections, we will learn how to perform multiple linear regression by using this function.

Perform Multiple Linear Regression

To perform multiple linear regression, please import the desired data, here will use three independents and one dependent.
1. Load data for multiple linear regression. All the independent data should be stored in a matrix, and dependent data in a vector.

```
// 1. get data for multiple linear regression
Worksheet wks = Project.ActiveLayer();
if( !wks )
    return; // please make sure a worksheet with data is active

DataRange dr;
dr.Add("X", wks, 0, 0, -1, 2);  // first three columns
dr.Add("Y", wks, 0, 3, -1, 3);  // the fourth column

matrix mX;
dr.GetData(mX, 0, 0);  // get data of first three columns to matrix

vector vY;
dr.GetData(&vY, 1);  // get data of the fourth column
```

2. Declare and initialize the parameters that will be passed to the function.

```
// 2. prepare input and output variables
UINT nOSizeN = mX.GetNumRows();  // number of observations
UINT nVSizeM = mX.GetNumCols();  // total number of independent
variables

LROptions stLROptions;  // use to set linear regression options
stLROptions.UseReducedChiSq = 1;

FitParameter stFitParameters[4]; // should be nVSizeM+1
UINT nFitSize = nVSizeM+1;  // size of FitParameter

RegStats stRegStats;  // use to get regression statistics
RegANOVA stRegANOV;  // use to get anova statistics
```

3. Pass the prepared parameters to the function and perform multiple linear regression.

```
// 3. perform multiple linear regression, here we are not going to get
// the covariance and correlation matrix of estimate, and no weight is
used.
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;
}
```

Output the Results

After finishing the calculation, the results may need to output to somewhere for presentation, such as Script Window, Result Log, Worksheet, etc.
Please refer to the **Result to Output Window** and **Result to Worksheet** section in the chapter **Analysis and Applications: Curve Fitting: Linear Fitting** for more details about how to output the results.

11.3.4 Non-linear Fitting

NLFit is the new fitter starting with Origin version 8. This new fitter handles the fitting process with a copy of the data while performing iterations. This results in faster operation compared to older versions where the fitter directly accessed data repeatedly from the worksheet.
There are two classes available to perform nonlinear fitting:

NLFit

This is the Origin C class that wraps the low level API from the new fitting engine. This class has no knowledge of Origin and works with copies of data in buffers. In order to use this class, you will be responsible in preparing all necessary buffers (pointers). This separation prepares for the future implementation of performing fitting as a background process.

NLFitSession

This is a higher level Origin C class with a friendly interface that wraps the NLFit class to Origin objects. This class is the kernel in the new NLFit Dialog. We recommend that you use this class in your Origin C code, as the process to interface to Origin is rather complicated and the NLFitSession classtakes care of this complexity for you.

Nonlinear Fitting

Before you use the NLFitSession class, you need to include a specific header file:

```
#include <..\originlab\NLFitSession.h>
```

You also need to include and compile the *OriginC\Originlab\nlsf_utils.c* file to your current workspace. Run the Labtalk command below in the Command window, or from your script file, to programmatically add the file:

```
Run.LoadOC(Originlab\nlsf_utils.c, 16)
```

Define an NLFitSession object, and set the fitting function as Gauss:

```
// Set Function
NLFitSession nlfSession;
if ( !nlfSession.SetFunction("Gauss") )
{
        out_str("Fail to set function!");
        return;
}
```

```
// Get parameter names and number:
vector<string>  vsParamNames;
int nNumParamsInFunction =
nlfSession.GetParamNamesInFunction(vsParamNames);
```

Set two XY datasets with DATA_MODE_GLOBAL mode, to perform global fitting with sharing of parameters:

```
int nNumData = 2;
// Set the first dataset
if ( !nlfSession.SetData(vY1, vX1, NULL, 0, nNumData) )
{
       out_str("Fail to set data for the first dataset!");
       return;
}

// Set the second dataset
if ( !nlfSession.SetData(vY2, vX2, NULL, 1, nNumData, DATA_MODE_GLOBAL)
)
{
       out_str("Fail to set data for the second dataset!");
       return;
}
```

Run parameter initialization code to initialize parameter values:

```
// Parameter initialization
if ( !nlfSession.ParamsInitValues() )
{
       out_str("Fail to init parameters values!");
       return;
}
```

Alternately, you can directly set parameter values one by one:

```
vector vParams(nNumParamsInFunction*nNumData);
// set parameter value for the first dataset
vParams[0] = 5.5; // y0
vParams[1] = 26; // A
vParams[2] = 8; // xc
vParams[3] = 976; // w

// set parameter value for the second dataset
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 means no error
    return;
```

Share xc parameter between the two datasets:

```
int nSharedParamIndex = 1; // 1, the index of xc in Gauss function
```

```
nlfSession.SetParamShare(nSharedParamIndex);
```

Perform the fit and output status message:

```
// Do fit
int nFitOutcome;
nlfSession.Fit(&nFitOutcome);

string strOutcome = nlfSession.GetFitOutCome(nFitOutcome);
out_str("Outcome of the fitting session : " + strOutcome);
```

Get fit statistic result:

```
int nDataIndex = 0;
RegStats        fitStats;
NLSFFitInfo     fitInfo;
nlfSession.GetFitResultsStats(&fitStats, &fitInfo, false, nDataIndex);
printf("# Iterations=%d, Reduced Chisqr=%g\n", fitInfo.Iterations,
 fitStats.ReducedChiSq);
```

Get final fit parameter values:

```
vector          vFittedParamValues, vErrors;
nlfSession.GetFitResultsParams(vFittedParamValues, vErrors);

// The parameter xc is shared in two input data.
// So the value of xc is same for all data sets, and it only appears
one time
// in the fitted parameter values - vParamValues.
// vsParamNames contains the parameter names in Gauss function - y0,
xc, w, A.
// The following to add parameter names for the second dataset without
xc.
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]);
}
```

Calculate fit curve Y values using the final fit parameters:

```
vector vFitY1(vX1.GetSize()), vFitY2(vX2.GetSize());
// Get fitted Y for the first dataset
nlfSession.GetYFromX(vX1, vFitY1, vX1.GetSize(), 0);
// Get fitted Y for the second dataset
nlfSession.GetYFromX(vX2, vFitY2, vX1.GetSize(), 1);
```

<u>Accessing FDF File</u>

Fitting function settings stored in an FDF file can be loaded into a tree variable. You will need to include the *OriginC\system\FDFTree.h* file:

```
#include <FDFTree.h>
```

Then use the *nlsf_FDF_to_tree* function:

```
string strFile = GetOpenBox("*.FDF");
Tree tr;
if(nlsf_FDF_to_tree(strFile, &tr))
{
        out_tree(tr);
}
```

11.3.5Find XY

The following procedure will show how to use the specified parameter values to get the value of the dependent variable from the independent variable or get the value of the independent variable from the dependent variable.

Linear

The formula of getting Y from X:

```
y = a + x * b;
```

The formula of getting X from Y:

```
x = (y - a) / b;
```

Non-linear

For non-linear function, we use NumericFunction class to get y from x and use ocmath_find_xs function to get x from y.

Get Y from X

```
#include <ONLSF.h>
#include <..\Originlab\nlsf_utils.h>
void _findy_from_x()
{
        // Please use the proper function from the related category in
        // Fitting Function Organizer dialog. Press F9 to open this
dialog.
        // The Poly function below under Polynomial category.
        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);
}
```

Get X from Y

The following function shows how to get two X values from the specified Y value. Before running, please import Samples\Curve Fitting\Gaussian.dat to Worksheet and keep the Worksheet active.

```
#include <...\originlab\nlsf_utils.h>
#include <FDFTree.h>
void _findx_from_y()
{
        double y = 20; // assign 20 as Y value to find X value

        Worksheet wks = Project.ActiveLayer();
        if (!wks)
        {
                return;
        }

        //get data to fitting
        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; //set how many x should be found
        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]);
}
```

11.4 Signal Processing

Origin C provides a collection of global functions and NAG functions for signal processing, ranging from smoothing noisy data to Fourier Transform (FFT), Short-time FFT(STFT), Convolution and Correlation, FFT Filtering, and Wavelet analysis. The Origin C functions are under the Origin C help -> Origin C Reference -> Global Functions -> Signal Processing category.

11.4.1Smoothing

The **ocmath_smooth** function support 3 methods: median filter, Savitzky-Golay smoothing and adjacent averaging smoothing.

```
vector vSmooth; // output
vSmooth.SetSize(vSource.GetSize());

//do Savitzky-Golay smoothing, 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);
```

11.4.2FFT

Before using fft_* functions, you need to include fft_utils.h.

```
#include <fft_utils.h>
```

FFT

fft_real performs a discrete Fourier transform(FFT_FORWARD) or inverse Fourier transform(FFT_BACKWARD).

```
fft_real(vSig.GetSize(), vSig, FFT_FORWARD); // return 0 for no error
```

Frequency Spectrum

fft_one_side_spectrum is used to compute the one side spectrum of FFT Result.

```
fft_one_side_spectrum(vSig.GetSize(), vSig); // return 0 for no error
```

IFFT

```
fft_real(vSig.GetSize(), vSig, FFT_BACKWARD); // return 0 for no error
```

STFT

The **stft_real** function is used to perform a Short-Time-Fourier-Transform on 1d signal real data. The **stft_complex** function is used to perform a Short-Time-Fourier-Transform on 1d signal complex data. The following is an example for real data.

```
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");
}
```

11.4.3FFT Filtering

Origin C supports multiple filter types for performing FFT Filtering, including: low pass, high pass, band pass, band block, threshold, and low pass parabolic. For example:

```
double dFc = 6.5;
int iRet = fft_lowpass(vecSignal, dFc, &vecTime);
```

11.4.4Wavelet Analysis

In Origin C, you can call a NAG function to do Wavelet analysis. To see all wavelet functions, go to the Origin C Help -> Origin C Reference -> Global Function -> NAG Functions -> Accessing NAG Functions Category and Help -> Wavelet category. It is necessary to include the related header.

```
#include <..\OriginLab\wavelet_utils.h>
```

The following is an example of a real type, one-dimensional, continuous wavelet transform.

```
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);
```

11.5 Peaks and Baseline

11.5.1Creating a Baseline

The **ocmath_create_baseline_by_masking_peaks** function can create a baseline according to only positive peaks, only negative peaks, or both direction peaks.
The following example shows how to create a baseline for the positive peaks and the negative peaks in input XY data(vx, vy).

```
// Allocate memory for baseline XY vectors
vector vxBaseline(vx.GetSize()), vyBaseline(vx.GetSize());

// find baseline XY data
int nRet = ocmath_create_baseline_by_masking_peaks(vx.GetSize(), vx, 
vy,
        vxBaseline.GetSize(), vxBaseline, vyBaseline, BOTH_DIRECTION);

// Ascending sort baseline XY data by X data.
if( OE_NOERROR == nRet )
{
        vector<uint> vn;
        vxBaseline.Sort(SORT_ASCENDING, true, vn);
        vyBaseline.Reorder(vn);
}
```

11.5.2Removing a Baseline

If the x coordinate of a baseline is the same as that of the peak curve, you can directly subtract, otherwise you need do interpolation before removing the baseline.
The following code shows how to do interpolation and then remove a baseline.
Assume the current worksheet has 4 columns in which to put peak XY data and baseline XY data.

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

Column colPeakX(wks, 0), colPeakY(wks, 1);
Column colBaseLineX(wks, 2), colBaseLineY(wks, 3);

// Get peak XY data.
// Get Y data by reference since want to subtract baseline on it below.
vector vPeakX = colPeakX.GetDataObject();
vector& vPeakY = colPeakY.GetDataObject();

// Get base line data
```

```
vector vBaselineX = colBaseLineX.GetDataObject();
vector vBaselineY = colBaseLineY.GetDataObject();

if( vPeakX.GetSize() != vPeakY.GetSize()
        || vPeakX.GetSize() == 0
        || vBaselineX.GetSize() == 0
)
        return;

// do interpolation on baseline data to keep x coordinate same as peak
data.
vector vyBaseTemp(vPeakX.GetSize());
if(OE_NOERROR != ocmath_interpolate(vPeakX, vyBaseTemp,
vPeakX.GetSize(),
        vBaselineX, vBaselineY, vBaselineX.GetSize(),
INTERP_TYPE_LINEAR))
{
        return;
}

// subtract base line
vPeakY -= vyBaseTemp;
```

11.5.3 Finding Peaks

The **ocmath_find_peaks_*** function is used to find peaks by multiple methods. The following example shows how to find a local maximum point in a local scope selected by nLocalPts. For a current point marked by nIndex, the scope is [nIndex-nLocalPts, nIndex+nLocalPts].

```
// Allocate memory for output vectors
UINT nDataSize = vxData.GetSize();
vector vxPeaks(nDataSize), vyPeaks(nDataSize);
vector<int> vnIndices(nDataSize);

// nDataSize, on input, the size of vxData, vyData;
// on output, return number of peaks
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 supports two functions: **ocmath_test_peaks_by_height** and **ocmath_test_peaks_by_number**, to verify peaks by specified height and peak number, respectively.
The following is an example showing how to verify peaks by minimum peak height.

```
// Get minimum and maximum from source Y data
```

```
double dMin, dMax;
vyData.GetMinMax(dMin, dMax);

// Get the bigger value from the highest point or the lowest point.
// And multiply 20% to get the peak minimum height.
double dTotalHeight = max(abs(dMax), abs(dMin));
double dPeakMinHeight = dTotalHeight * 20 / 100;

// Verify peaks by specified minimum height
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]);
}
```

11.5.4 Integrating and Fitting Peaks

Integrate Peak

The **ocmath_integrate** function is used to integrate to find the area under a curve. The following example shows how to perform integration on the sub curve of one peak.

```
int i1 = 51, i2 = 134; // From/to index to set the sub range of one
peak
IntegrationResult IntResult; // Output, integration result
vector vIntegral(i2+1); // Output, integral data

// Integrate and output result
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);
}
```

Fitting Peak

The Origin C **NLFitSession** class supports a method to fit peaks with a different fitting function.
Refer to the **Curve Fitting** chapter to see a more in depth description and examples about this class.

11.6 Using NAG Functions

11.6.1Header Files

To call any NAG function, you need to include the header file or files where the NAG function is declared.
A single header file, which includes all the commonly used NAG header files, is provided below. Usually, you can just include this header file in your code.

```
#include <OC_nag8.h> // includes all common NAG header files
```

If only a single NAG function or just a few are used, you can also just include its (their) own individual NAG header file(s). For example, if the NAG function *f02abc* is called in the code, then two related NAG header files need to be included.

```
#include <NAG8\nag.h>    // NAG struct and type definitions
#include <NAG8\nagf02.h> // contains the f02 function declarations
```

11.6.2Error Structure

All NAG functions accept one argument, which is a pointer of NagError structure. This structure is used to test whether the NAG function is executing successfully or not. The example below shows whether the NAG function *f02abc* works successfully.

```
NagError err;                    // Declare an error structure
f02abc(n, mx, n, r, v, n, &err); // Call NAG f02abc function
if( err.code != NE_NOERROR )     // If an error occurred
    printf(err.message);         // Output error message
```

If you don't need to know whether the call is successful or not, the error structure declaration is not needed. And the **NAGERR_DEFAULT** macro can be passed instead. This macro is a NULL pointer. To ensure compatibility with future versions of NAG functions, it will be better to use this macro if you can work without error structure.

```
f02abc(n, mx, n, r, v, n, NAGERR_DEFAULT);
```

11.6.3Callback Functions

In the NAG Library, most of the routines involve callback functions. Before defining a callback function, you need to know the return type and argument types of the callback function that NAG will expect when calling it.
Take the NAG function *d01ajc* for example. In the header file *nagd01.h*, we can see that the first argument is **NAG_D01AJC_FUN f**. This argument is a callback function. Then in *nag_types.h*, we find that **NAG_D01AJC_FUN** is a type of **NAG_D01_FUN**, which is defined as:

```
typedef double (NAG_CALL * NAG_D01_FUN)(double);
```

Then we can define the callback function as follows:

```
double NAG_CALL myFunc(double x)
{
    double result;
    // Do processing on 'x'
    return result;
}
```

When calling the NAG function *d01ajc*, *myFunc* (defined above) can be passed as the first argument.

Calling c05adc Example

This example will show how to call the NAG function *c05adc*, the fourth argument of which is the callback function argument. This callback function of type **NAG_C05ADC_FUN** is defined in *nag_types.h*.

```
typedef double (NAG_CALL * NAG_C05ADC_FUN)(double);
```

From the definition, we know that both the return type and the only argument type are double. So we define the callback function as follows:

```
double NAG_CALL myC05ADCfunc(double x)
{
    return exp(-x)-x;
}
```

The following code shows how to call the *c05adc* function by passing the *myC05ADCfunc* callback function.

```
double a = 0.0, b = 1.0, x, ftol = 0.0, xtol = 1e-05;
NagError err;

c05adc(a, b, &x, myC05ADCfunc, xtol, ftol, &err);
```

11.6.4NAG Get Data From Origin

Many NAG functions take a pointer to an array of numeric data. Both Origin worksheets and matrix sheets allow getting a pointer to their data. This pointer can be passed to NAG functions. In Origin C, data is commonly passed using Dataset or DataRange objects. The sections below will show how to pass data from a worksheet by using Dataset and DataRange. The DataRange way is recommended.

Dataset

A Dataset object can be passed to a NAG function as long as the Dataset is of the data type expected by the NAG function. The data type of an Origin worksheet column is **Text & Numeric** by default. For most, but not all, NAG functions, this data type is not allowed to be passed, because NAG functions expect floating or integer pointers.
If you make sure that the Dataset is of the type expected by the NAG function, the following code can be used to pass a Dataset object to a NAG function.

```
// Get access to the active worksheet.
Worksheet wks = Project.ActiveLayer();

// Construct Datasets to get access to the wks data.
Dataset dsX, dsY;
dsX.Attach(wks, 0);
dsY.Attach(wks, 1);

// Call NAG's nag_1d_spline_interpolant(e01bac) function.
NagError err;
Nag_Spline spline;
e01bac(m, dsX, dsY, &spline, &err);
```

DataRange

The DataRange class provides the GetData method for getting data from a worksheet into a vector, even if the worksheet columns are of the **Text & Numeric** data type. The GetData method can also ignore the rows with missing values easily, which is very important when passing data to NAG functions.
Using DataRange to pass data from Origin to NAG functions is much safer, and is recommended. The following example demonstrates how to do that.

```
void call_NAG_example()
{
        int i, numPoints = 5;

        // Create a new worksheet page.
        WorksheetPage pg;
        pg.Create("origin");

        // Get access to the active worksheet and add two more columns.
        Worksheet wks = Project.ActiveLayer();
        // Add X2 column
        i = wks.AddCol();
        Column col(wks, i);
        col.SetType(OKDATAOBJ_DESIGNATION_X);
        // Add Y2 column
        wks.AddCol();

        // Create some starting XY values in first two columns
        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));
        }

        // Create data range object.
        DataRange dr;
        dr.Add(wks, 0, "X");
        dr.Add(wks, 1, "Y");

        // Copy data from wks to vector using data range.
        // This copy will ignore rows with missing values.
        vector vX1, vY1;
        dr.GetData(DRR_GET_DEPENDENT, 0, NULL, NULL, &vY1, &vX1);

        // Call NAG to calculate coefficients.
        NagError err;
        Nag_Spline spline;
        e01bac(vX1.GetSize(), vX1, vY1, &spline, &err);

        // Get the spline's XY values
        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);
                }
        }

        // Free memory allocated by NAG
        NAG_FREE(spline.lamda);
        NAG_FREE(spline.c);

        // Copy spline values to worksheet
        dsX.Attach(wks, 2);
        dsX = vX2;

        dsY.Attach(wks, 3);
        dsY = vY2;
}
```

11.6.5 How to Call NAG e04 Functions

NAG e04 functions are mainly used for minimizing or maximizing a function. All these e04 functions need a parameter, which is a pointer to **Nag_E04_Opt** structure. After executing this kind of functions, the results will output to the windows console by default. However, this behavior is not safe when running in Origin. So, the default pointer, **E04_DEFAULT**, can not be used here, it needs to initialize the **Nag_E04_Opt** structure variable first, and then change the output target to a file before passing to the NAG function.
The following example will show how to call the NAG function, **nag_opt_simplex**, safely. And the results will output to a file.

```
#include <OC_nag8.h>
void text_e04ccc()
{
        double objf;
        double x[2];
        Integer n;

        printf("\ne04ccc example: \n");

        NagError fail;  // error
        Nag_E04_Opt opt;  // e04 optional parameter

        nag_opt_init(&opt);  // initialize optional parameter, e04xxc =
nag_opt_init

        // change output target to a file, the result will be in this
file
```

```
        strcpy(opt.outfile, "C:\\result.txt");

        n = 2;
        x[0] = 0.4;
        x[1] = -0.8;
        try
        {
                // call the NAG function, e04cccc = nag_opt_simplex
                nag_opt_simplex(n, funct, x, &objf, &opt, NAGCOMM_NULL,
&fail);
        }
        catch(int err)
        {
                printf("\nerror = %d\n", err);  // if there is an
exception
        }
        printf("fail->code = %d\n", fail.code);  // error code
        printf("fail->message = %s\n", fail.message);  // error message
}

// call back function for nag_opt_simplex
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);
}
```

12 Output Objects

12.1 Results Log

The Results Log is an output window that automatically stamps each block of output with the date and time and the name of the window associated with the results. The user interface allows users to configure which results are displayed and to float the window or dock it to Origin's main window.
The following example shows the simplest way to output to the Results Log from Origin C using the *OutStringToResultsLog* method of the Project class. While this is the simplest way to output to the Results Log, it can also be considered the most limited. Each call to the *OutStringToResultsLog* method is considered an individual log and will be stamped with the current date and time and the name of the associated window.

```
string str = "Column1\tColumn2\tColumn3\n3.05\t17.22\t35.48";
Project.OutStringToResultsLog(str);
```

12.2 Script Window

The Script Window is the default output window for Origin C. Whenever you output strings or numeric values they will appear in the Script Window. You can change which window such output appears in by setting LabTalk's Type.Redirection property. This property allows you to redirect your application's output to the Script window, Command window, Results Log, or even a Note window. See LabTalk's *Type.Redirection* property for more details.
The following example will save the current Redirection setting, set it to redirect output to the Script window output, then the Command window, and then restore the saved setting.

```
string strTypeRedir = "type.redirection";
double dCurTypeRedir;
LT_get_var(strTypeRedir , &dCurTypeRedir); // get current

LT_set_var(strTypeRedir , 5); // 5 for Script window
out_str("Hello Script Window");

LT_set_var(strTypeRedir , 128); // 128 for Command window
out_str("Hello Command Window");

LT_set_var(strTypeRedir , dCurTypeRedir); // restore current
```

The next example will use *Type.Redirection* and *Type.Notes$* to redirect Origin C output to a Note window.

```
Note note;
note.Create(); // Create the target Note window
```

```
LT_set_str("type.notes$", note.GetName());
LT_set_var("type.redirection", 2); // 2 for Note window

printf("Hello Notes Window");
```

12.3 Notes Window

The first example shows how to work with the text of a Note window using the Text property. The Text property is a string class type which allows you to use all the string capabilities of strings.

```
Note note;
note.Create(); // Create the target Note window
if( note )
{
    note.Text = "Hello Note window.";
    note.Text += "\nAnother line of text."
}
```

The next example will use Type.Redirection and Type.Notes$ to redirect Origin C output to a Note window.

```
Note note;
note.Create(); // Create the target Note window

LT_set_str("type.notes$", note.GetName());
LT_set_var("type.redirection", 2); // 2 for Note window

out_str("Hello Notes Window");
```

12.4 Report Sheet

The Datasheet class has the **GetReportTree** and **SetReportTree** methods for getting and setting a report into a worksheet or matrix sheet.

```
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.");
```

13 Accessing Database

13.1 Importing from a Database

Origin C includes the ability to import data from a database into a worksheet. The following example shows how to do this by importing an Access database file, included in the Origin Samples folder. An *ADODB.Reocrdset* object can refer to MSDN. To find out how to construct your connection string, refer to DbEdit X-Function

```
Object ocora;

try
{
        ocora = CreateObject("ADODB.Recordset");
}
catch(int nError)
{
        out_str("Failed to create ADODB.Recordset");
        return FALSE;
}

// Import stars.mdb from the Origin Samples folder
string  strDatabaseFile = GetAppPath(1) +
    "Samples\\Import and Export\\stars.mdb";

// Prepare the database connection string
string strConn;
strConn.Format("Provider=Microsoft.Jet.OLEDB.4.0; Data Source=%s;
    User ID=admin; Password=;", strDatabaseFile);

// Prepare the SQL string
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();

//put data into the worksheet.
BOOL                   bRet = wks.PutRecordset(ocora);
```

```
out_int("bRet = ", bRet);
return bRet;
```

13.2 Exporting into a Database

Origin C has the ability to export data from a worksheet to a specified database table. The following steps show how to export fitting summary data into a database.

1. Set up a database named "Analysis" in MySQL, and assume it is running on the machine "Lintilla".
2. Create a table named "FittingSummary" with 9 fields, set the data type of the first two fields as varchar(40) and the rest as double.
3. Open OriginExe\Samples\Curve Fitting\autofit.ogw, and fill the columns on the "Data" layer with data.
4. After recalculating, activate the "Summary" layer, and run the following code to export the result to a database.

```
//user should modify connect and query string according to their
database settings.
//such as value of Server, Database, UID, PWD etc.
#define STR_DB_CONN                           "Driver={MySQL ODBC 3.51
Driver};       \

        Server=Lintilla;Port=3306;Option=4;Database=Analysis;UID=test;PW
D=test;"
#define STR_QUERY                             "Select * from
FittingSummary"

bool    write_wks_to_db()
{
        Worksheet wks = Project.ActiveLayer();
        if ( wks )
                return false;

        //connect to database "Analysis" on "Lintilla"
        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!");

        //open recordset
        oRecordset.CursorLocation = 3; //adUseClient, please refer to 
MSDN for details
        oRecordset.Open(strQuery, oConn, 1, 3); //adOpenKeyset, 
adLockOptimistic

        int iRowBegin = 0, nRows = 8; //8 rows
        int iColBegin = 0, nCols = 9; //9 columns

        //LAYWKSETRECORDSET_APPEND for appending new recordset;
        //LAYWKSETRECORDSET_REPLACE for replacing existing recordsets.
        int nOption = LAYWKSETRECORDSET_APPEND; //append.

        int nRet = wks.WriteRecordset(oRecordset, nOption,
                        iRowBegin, nRows, iColBegin, nCols);
        return (0 == nRet);
}
```

13.3 Accessing SQLite Database

SQLite is a software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine, and it has become the most widely deployed SQL database in the world. Due to the excellent features, SQLite is now widely used in different applications and systems.
SQLite3 is the latest version released. Origin provides a DLL for accessing SQLite database from Origin C. It is necessary to include a header file that contains the prototypes of SQLite3 APIs:

```
#include <oc_Sqlite.h>
```

A simple example of how to use these functions is available at the end of the header file.
Origin C also provides a wrapped class, **OSQLite**, that makes accessing SQLite much easier. To use this Origin C class, the header file containing this class must be included, like:

```
//DataSet1.1.db is a database file, which contains a table named 
Originlab
//The table is created with the following statement
//CREATE TABLE OriginLab(ID INTEGER NOT NULL, NUMBER INTEGER NOT NULL, 
SALARY INTE
//GER NOT NULL, Data BLOB NOT NULL);
#include <..\Originlab\oSQLite.h> //required header file
#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);
        //after modify the data, may use the following code to export 
data
        //sqlObj.Export("OriginLab", wks);
}
```

14 Accessing LabTalk

Origin C has the ability to get and set LabTalk numeric and string values and run LabTalk scripts.

14.1 Getting and Setting LabTalk Numeric Values

The Origin C **LT_get_var** and **LT_set_var** global functions are used for getting and setting LabTalk numeric values. Numeric values include variables, system variables and object properties.

```
double dOriginVer;
LT_get_var("@V", &dOriginVer);
printf("Running Origin version %f\n", dOriginVer);
```

This is how to set the minimum font size used in the Data Display window.

```
LT_set_var("System.DataDisplay.MinFontSize", 12);
```

There are times when you will want to temporarily set a LabTalk variable, do some work, and then restore the LabTalk variable to its original value. There are mainly two ways to do this. The first way is the long way to do it using *LT_get_var* and *LT_set_var*.

```
double dProgressBar;
LT_get_var("@NPO", &dProgressBar); // get starting value
LT_set_var("@NPO", 0); // set new value
//
// do some work
//
LT_set_var("@NPO", dProgressBar); // restore starting value
```

The next way is the simple way using the *LTVarTempChange* class. To use the class you simply pass the variable name and the temporary value. The constructor saves the starting value into a data member and sets the variable to the temporary value. The destructor will restore the variable to its starting value.

```
{
    LTVarTempChange progressBar("@NPO", 0);
    //
    // do some work
    //
}
```

14.2 Getting and Setting LabTalk String Values

The Origin C **LT_get_str** and **LT_set_str** global functions are used for getting and setting LabTalk string values. String values include variables, string substitution variables and object properties.

```
char szCustomDateFmt[200];
```

```
LT_get_str("System.Date.CustomFormat1$", szCustomDateFmt, 200);
printf("Custom Date Format 1:  %s\n", szCustomDateFmt);
```

This is how to set the font used in the Data Display window.

```
LT_set_str("System.DataDisplay.Font$", "Courier");
```

This is how to rename the active sheet of the active book.

```
LT_set_str("wks.name$", "MySheet");
```

14.3 Running LabTalk Script

The Origin C **LT_execute** global function allows you to run LabTalk script stored in a string. The Format string method can be useful in passing Origin C variables into the LabTalk script:

```
string strScript;
string strBook = "Book1";
int iColStart = 2, iColEnd = 5;
strScript.Format("win -a %s;plotxy %u:%u;", strBook, iColStart,
iColEnd);
LT_execute(strScript);
```

The next example calls the *LT_execute* method of the Layer class instead of the global *LT_execute* function. When calling the global *LT_execute* function, the script runs and will operate on the active layer when no layer is specified by the script code. When calling the *LT_execute* method of the Layer class, the script runs and will operate on the layer instance instead of the active layer.

```
WorksheetPage wksPg("Book1");
Worksheet wks = wksPg.Layers(0);

WorksheetPage wksPgActive;
wksPgActive.Create("Origin"); // This page is now active

LT_execute("wks.colWidth=16");    // Set column widths of active layer
wks.LT_execute("wks.colWidth=8"); // Set column widths of Book1
```

14.4 Embedding LabTalk Script in Origin C Code

LT_execute allows you to execute the LabTalk script contained in a string, but there are times when you will want to execute a large block of script that you may not want to put into a string, for readability. For those times you can use the *_LT_Obj* block. The *_LT_Obj* block allows you to embed a large block of LabTalk script code right into the flow of your Origin C code to access LabTalk objects. For LabTalk objects, please refer to LabTalk Help: LabTalk Programming: Language Reference: Object Reference

```
out_str("Choose an image file...");

_LT_Obj // Use LabTalk's FDlog to show a file dialog
{
        // Origin C code
        string strDefaultPath = GetOriginPath(); // to get Origin EXE
path

        // LabTalk script to access FDLog object
        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);
```

15 User Interface

This chapter demonstrates ways in which Origin C functions allow user interaction.

15.1 Wait Cursors

The *waitCursor* class changes the mouse pointer to the hour glass, or busy indicator. It is a visual cue to indicate that Origin is running a piece of code that may require an amount of time sufficiently large that it will be prevented from responding to other requests. The mouse pointer changes to the busy indicator when an instance of a *waitCursor* object is created, and changes back to the arrow when the instance is destroyed.
The following example is a function that pretends to do something time consuming. At the beginning, we declare and create a *waitCursor* instance. During the creation, the mouse pointer will be changed to the busy indicator. When the function exits, the *waitCursor* instance is automatically destroyed, causing the mouse pointer to be changed back to the arrow.

```
void myTimeConsumingFunction()
{
    waitCursor wc; // declare and show the wait cursor
    for( int i = 0; i < 10000; i++ )
    {
        if( 0 == (i % 100) )
            printf("i == %d\n", i);
    }
}
```

The next example is similar to the above example, but adds the ability to exit the function before it finishes its time consuming task. The exiting early ability is accomplished by calling the wait cursor's *CheckEsc* method. This method returns true if the user has pressed the Esc key, otherwise it returns false.

```
void myEscapableTimeConsumingFunction()
{
    waitCursor wc; // declare and show the wait cursor
    for( int i = 0; i < 10000; i++ )
    {
        if( 0 == (i % 100) )
            printf("i == %d\n", i);
        if( wc.CheckEsc() )
            break; // end loop early
    }
}
```

15.2 Common Dialog Boxes

15.2.1 Input Box

Input boxes are used to solicit textual information from program users. The global function *InputBox* is used to open an input box.

```
// enter string
string strName = InputBox("Please enter your name", "");
printf("Name is %s.\n", strName);

// enter numeric
double dVal = InputBox(0, "Please enter a value");
printf("Value is %g.\n", dVal);
```

15.2.2 Message Box

Message boxes are used to convey information, or to prompt a user with a limited number of choices. The information presented is considered important enough to require the user's attention before they are allowed to continue.
The first example shows a simple OK message box to inform the user that their file has downloaded successfully.

```
string strTitle = "File Download";
string strMsg = "Your file downloaded successfully.";
MessageBox(GetWindow(), strMsg, strTitle, MB_OK);
```

The next example shows an OK-Cancel message box with an exclamation icon to warn the user that they will not be able to undo an action. This gives the user a choice to proceed or to cancel their action.

```
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");
```

The next example shows a Yes-No message box with a question mark icon. This is being used to ask the user if they want to continue with their action.

```
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.");
```

15.2.3 Progress Box

A progress box is a small dialog box that indicates the software is busy processing data. This dialog box contains a progress bar for showing the fraction of the completed processing. The progress dialog box is usually used in iterative loops.

```
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!"); // Click Cancel button to abort
        break;
    }
    LT_execute("sec -p 0.5");
}
```

15.2.4File Dialogs

Origin C provides functions for all the common file and path dialogs. This includes dialogs that prompt the user to open a single file, open multiple files, save a file, and choose a folder. The following sections show you how to use these dialogs in your own applications.

File Open Dialog

```
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);
```

Multiple Files Open Dialog

```
StringArray saFilePaths;
StringArray saFileTypes(3);
saFileTypes[0]="[Project (*.OPJ)] *.OPJ";
saFileTypes[1]="[Old version (*.ORG)] *.ORG";
saFileTypes[2]="[Worksheets (*.OGW)] *.OGW";

// Press Ctrl or Shirt key to choose multiple files
int iNumSelFiles = GetMultiOpenBox(saFilePaths, saFileTypes,
GetAppPath(false));
```

File SaveAs Dialog

```
string strDefaultFilename = "Origin";
FDLogUseGroup nFDLogUseGroup = FDLOG_ASCII; // ASCII file group

string strPath =
GetSaveAsBox(nFDLogUseGroup,GetAppPath(false),strDefaultFilename);
out_str( strPath );
```

Path Browser Dialog

```
string strPath = BrowseGetPath(GetAppPath() + "OriginC\\", "This is an
example");
out_str(strPath);
```

15.3 GetN Dialogs

Origin C's *GetNBox* function is an easy way to show a full-featured dialog box. The *GetNBox* function uses a Tree that has been previously created using the GETN macros. The Tree contains nodes and branches containing more nodes. Each node represents an item in the dialog box. An item can be a check box, an edit box, a combo box, or another supported control. The following sections explain the steps involved in making a tree that can be passed to the *GetNBox* function.

15.3.1The GETN Macros

The *GetNBox* function requires a Tree containing information about what dialog it should create and how it should be presented. Although it requires a tree, the caller does not have to know much about trees, thanks to the GETN macros. The entire tree is created using the GETN macros.

The GETN_BOX and GETN_TREE macros

The first macro you will use when creating your tree is either the GETN_BOX or the GETN_TREE macro. The GETN_BOX macro constructs a base GetNBox tree and gives the dialog box the simple dialog appearance. The GETN_TREE macro constructs a base GetNBox tree and gives the dialog box a tree appearance. Both macros declare a new tree instance with the name passed to the macro. After declaration it will set up the base of the tree for the style requested.

```
GETN_BOX(trMyDlg)
GETN_TREE(trMyDlg)
```

Data Item Macros

After you have declared and set up your tree using the GETN_BOX or GETN_TREE macro you can start adding data item nodes. There are macros for adding string or numeric edit boxes, check boxes, drop down lists, button controls, sliders, and more.

```
#include <GetNbox.h>
void GETN_ex1()
{
    GETN_TREE(testTree)
        GETN_STR(name, "Name", "Jacky") // string edit box
        GETN_NUM(age, "Age", 31)    // numeric edit box
        GETN_LIST(gender, "Gender", 0, "Male|Female") // drop-down list
    GETN_BUTTON(path, "File Path", GetAppPath()) // button control
        GETN_OPTION_EVENT(button_event) // the event function of button
control

    if(GetNBox(testTree, NULL, NULL, NULL, NULL))
          out_tree(testTree);
}

// define the event function of button control
bool button_event(TreeNode& myTree, int nRow, int nType, Dialog&
theDlg)
{
        if(TRGP_STR_BUTTON == nType && nRow >= 0)
        {
           string strPath = BrowseGetPath(myTree.path.strVal);
           myTree.path.strVal = strPath;
           return true;
        }
        else
           return false;
}
```

The Branch Macros

A GetNBox can contain many items. To help keep items organized, there are the GETN_BEGIN_BRANCH and GETN_END_BRANCH macros. The begin branch macro starts a new branch that can contain child nodes. All data item macros that follow a begin branch will be children of that branch. The branch is ended when the end-branch macro is called.

```
#include <GetNbox.h>
void GETN_ex2()
{
    GETN_TREE(testTree)
        GETN_BEGIN_BRANCH(Personal, "Personal Info")
            GETN_STR(name, "Name", "") // string edit box
            GETN_NUM(age, "Age", 31)   // numeric edit box
            GETN_LIST(gender, "Gender", 0, "Male|Female") // drop-down
list
        GETN_END_BRANCH(Personal)

    if(GetNBox(testTree, NULL, NULL, NULL, NULL))
          out_tree(testTree);
}
```

Options Control

There are GetN macros to set format on GetN controls. For example, **GETN_OPTION_BRANCH** can control whether the branch default is open or closed.

```
GETN_BEGIN_BRANCH(Personal, "Personal Info")
    GETN_OPTION_BRANCH(GETNBRANCH_OPEN) // open the branch
    //...
GETN_END_BRANCH(Personal)
```

Apply Button

The default GetN Dialog has OK and Cancel buttons; the Apply button is optional. When the Apply button is displayed, and the user clicks this button, you may want to call the related event function to do something.
The following is an example showing how to display the Apply button on a GetN dialog, and call the event function **_apply_event** when the Apply button is clicked.

```
#include <GetNbox.h>
// The interface of apply button event function need to according to
// PAPPLY_FUNC typedef.
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;
}

void GETN_Apply_ex1()
{
    GETN_TREE(tr)
    GETN_COLOR(LineColor, "Color", 3)
    // the option to set color list to contain custom panel
    GETN_COLOR_CHOICE_OPTIONS(COLORLIST_CUSTOM | COLORLIST_SINGLE)

    bool bShowApplyButton = true;
    if(GetNBox(tr, NULL, "Example", NULL, GetWindow(),
bShowApplyButton,
                _apply_event))
    {
        out_str("Click OK");
    }
}
```

15.3.2Event Handling

Above, we have introduced the apply button event function, but GetN dialogs also support a mechanism to respond to other events, like dialog initialization, control value changes, and so on.
The following is an example showing how to access these types of event.

```
#include <GetNbox.h>
int _normal_event(TreeNode& tr, int nRow, int nEvent, DWORD& dwEnables,

        LPCSTR lpcszNodeName, WndContainer& getNContainer, string&
strAux,
        string& strErrMsg)
{
        if( 0 == lstrcmp(lpcszNodeName, "list") || GETNE_ON_INIT ==
nEvent )
        {
                tr.name.Enable = (1 == tr.list.nVal);
                tr.id.Show = (2 == tr.list.nVal);
        }
        return 0;
}

void GETN_Event_Ex()
{
    GETN_TREE(tr)
    GETN_LIST(list, "Options", 0, "None|Name|ID")
    GETN_STR(name, "Name", "")
    GETN_STR(id, "ID", "")

    if(GetNBox(tr, _normal_event, "Example", NULL, GetWindow()))
    {
        out_str("Click OK");
    }
}
```

15.4 Picking Points from a Graph

The Origin C **GetGraphPoints** class is used to pick points from the curve on the Graph window. This class has virtual methods to allow the user to derive from it to overload methods.
The following example shows how to use the *GetGraphPoints* class to pick two points from a Graph.

```
GetGraphPoints mypts;

// Set as true , the cursor moves along the DataPlot and picks points
from
// the curve.
// Set as false, the cursor does not move along the DataPlot, and picks
// points from the screen.
mypts.SetFollowData(true, dp.GetIndex());

// To pick point from the specified Graph by GraphLayer object(gl)
int nPts = 2; // the number of the points to pick
mypts.GetPoints(nPts, gl);

// Get the x/y data and indices from the picked points
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);
        }
}
```

15.5 Adding Controls to a Graph

If you want to attach a dialog to a page, similar to a workbook's organizer or the top of a polar graph, then the **SetSplitters** method of the **PageBase** class can be used to accomplish this.
To add a dialog bar to a page, **lpcszString** needs to include the dialog class name and the position (Left, Right, Top or Bottom) of the page window. Set **lpcszString** as NULL to remove the existing dialog bar.
The following example shows how to add and remove user-created dialog on a Graph window.
The class of the user-defined dialog:

```
#include <..\Originlab\DialogEx.h>
```

```
// OC_REGISTERED key word must allow the PageBase::SetSplitters method
// to find this class.
class OC_REGISTERED MyGraphPolarBar : public Dialog
{
public:
        // IDD_POLAR_CONTROL is dialog resource ID
        // Odlg8 is the name of dialog resource DLL file, if not
specified path,
        //default path is \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;
        }
};
```

Add or remove dialog on a Graph window.

```
void Page_SplittersControl(BOOL bShow = TRUE, int nPos = 2)
{
        Page pg = Project.Pages("Graph1");

        if( bShow )
        {
                int nPercent = 30;
                string strDlgClass = "MyGraphPolarBar"; // the above
dialog class

                string  strConfig;
                switch(nPos)
                {
                case 0: // Bottom
                        strConfig.Format("r{%s}r[%s]",
(string)nPercent+"%", strDlgClass);
                        break;
                case 1: // Right
                        strConfig.Format("c{%s}c[%s]",
(string)nPercent+"%", strDlgClass);
                        break;
                case 2: // Top
                        strConfig.Format("r[%s]{%d}r", strDlgClass,
nPercent);
                        break;
                case 3: // Left
                        strConfig.Format("c[%s]{%d}c", strDlgClass,
nPercent);
                        break;
                }
                pg.SetSplitters(strConfig);
        }
        else
                pg.SetSplitters(NULL); // remove dialog bar from page

}
```

16 Dialog Builder

Dialog Builder refers to the Origin C support to use Microsoft Visual C++ generated resource DLL for building floating tools, dialog boxes, and wizards in Origin. All resource elements, including Windows common controls and Origin's worksheet and graph controls can be accessed and controlled from Origin C. This capability used to require a Dialog Builder license, but since Origin 8.5, this restriction has been removed and all Origin installations include this support.
This guide contains a tutorial that shows how to use Microsoft Visual C++ to create a resource-only DLL containing a dialog and then how to use Origin C to display the dialog. Additional sections describe in more detail how to create a resource-only DLL and access it's resources from Origin C.

Developer Kit Samples

Developer Kit sample files are available as an Origin Package file (.OPX). This OPX file is available to our maintenance customers. Please contact OriginLab or your Origin distributor to obtain the file.
The OPX file will add all sample files, including resource DLLs, in the *\Samples\DeveloperKit\* subfolder under the Origin installation folder.

16.1 Simple Hello World Dialog

16.1.1Create Resource DLL in VC

Create by Origin Dialog AppWizard

1. Start Visual C++ 6.0, select File->New to open the New dialog. On the Projects tab, choose Origin Dialog AppWizard, set Project name to "ODialog", choose a Location and click OK.
2. Choose a simple dialog, and click Next.
3. Keep Origin C selected and click Finish, then click OK. The resource file with one simple dialog and the related source file and header file will be generated.
4. Click menu Build->Set Active Configuration to choose Debug or Release.
5. Choose menu Build->Builder ODialog.dll to create DLL.
6. Go to the file location specified above. Copy the DLL file to outside the Debug or Release folder, to keep the path of the DLL file the same as that of the ODialog.cpp file.
7. Open the ODialog.cpp file in Origin C Code Builder, compile, and run the DoMyDialog function to open the dialog.

Create by Win32 Dynamic-Link Library

This section describes how to create a Resource-only DLL in Visual C++ 6.0.

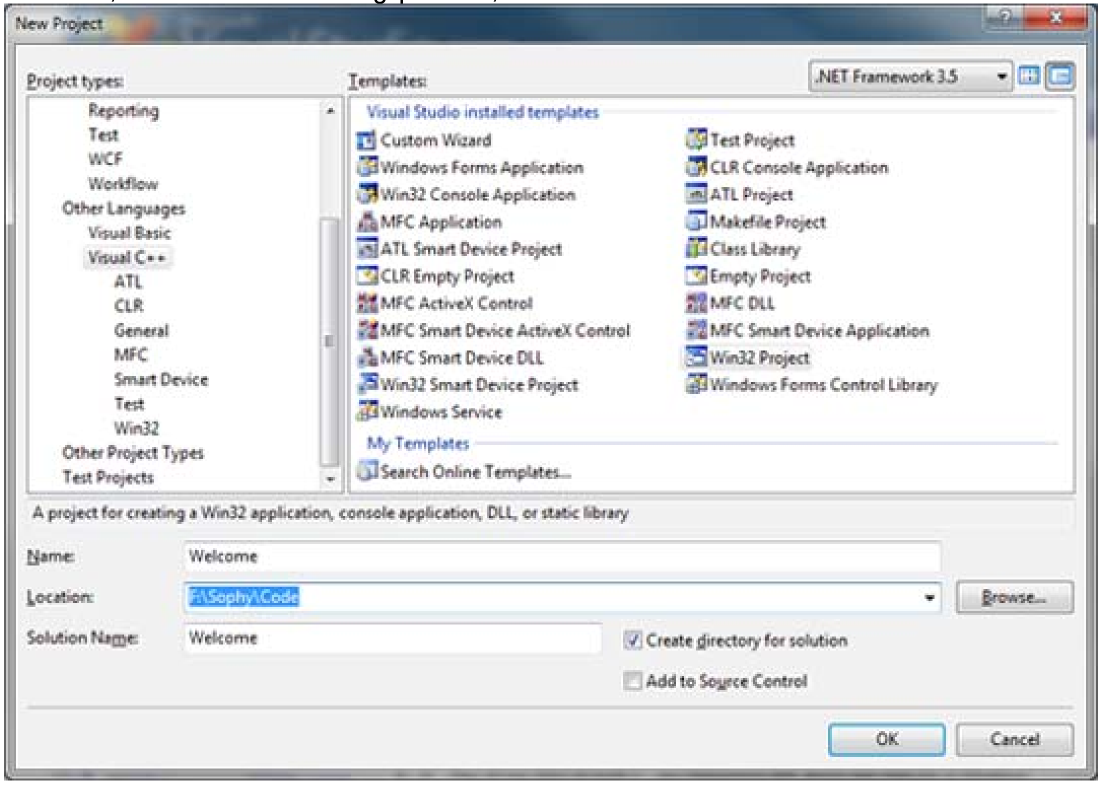
1. Start Visual C++ 6.0, select File->New to open the New dialog. On the Projects tab, choose Win32 Dynamic-Link Library as the project template, set the Project name as ODialog, choose a Location and click OK. In the dialog that appears, select a simple DLL project and click Finish.

2. Select Project->Settings to open the Project Settings dialog. On the Resources tab, set the Resource file name, like ODialog.res, and select the Language according to your software settings as English (United States), and click OK.
3. Select Insert->Resource to insert resources into the project. For a Dialog and controls on it, set dialog ID to IDD_OC_DIALOG.
4. Choose File->Save As to save the Resource Script file as ODialog.rc. Choose Project->Add To Project->Files, then choose the ODialog.rc file to add it to the project.
5. If the Language is not English, please do this step. In the Workspace view Resource tab, open the list tree, right click on IDD_OC_DIALOG, choose Properties, and then in the dialog choose Neutral for Language.
6. Build the whole project with Debug or Release configuration. The resulting DLL file is generated under the Debug or Release subfolder.

<u>**Create Resource-only DLL in Visual Studio 2008**</u>

This article describes in detail the general process of creating a Resource-only DLL in Visual Studio 2008. The following steps show how to build a Resource-only DLL with VS2008 that is accessible in Origin using Origin C.

1. Start Microsoft Visual Studio 2008.
2. Select File->New->Project to create a new project.
3. In the New Project dialog, choose Visual C++ as the programming language and Win32 Project as the template, type in the project name as "Welcome", select its location, like in the following picture, and click OK.



.NET Framework 3.5

A project for creating a Win32 application, console application, DLL, or static library

Name: Welcome

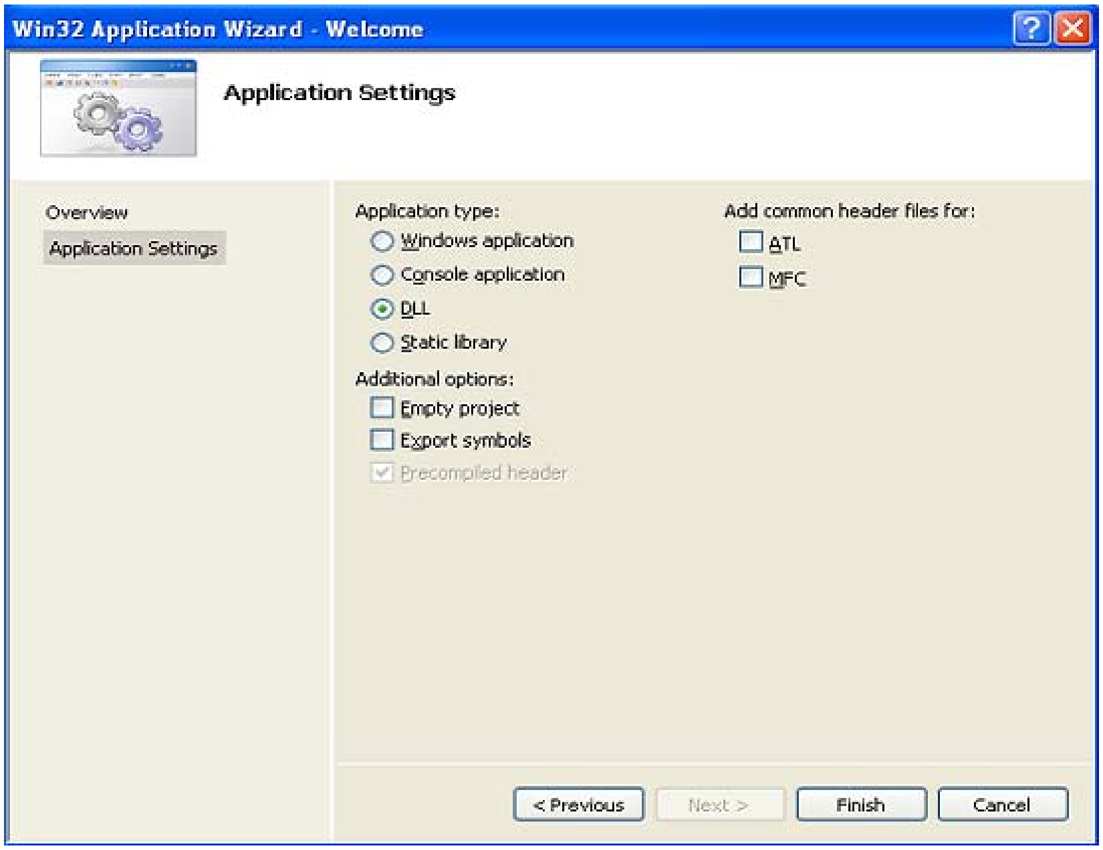
Location: F:\Sophy\Code

Solution Name: Welcome

[✓] Create directory for solution

[ ] Add to Source Control

4. In the Win32 Application Wizard dialog, set Application type as DLL and click Finish.

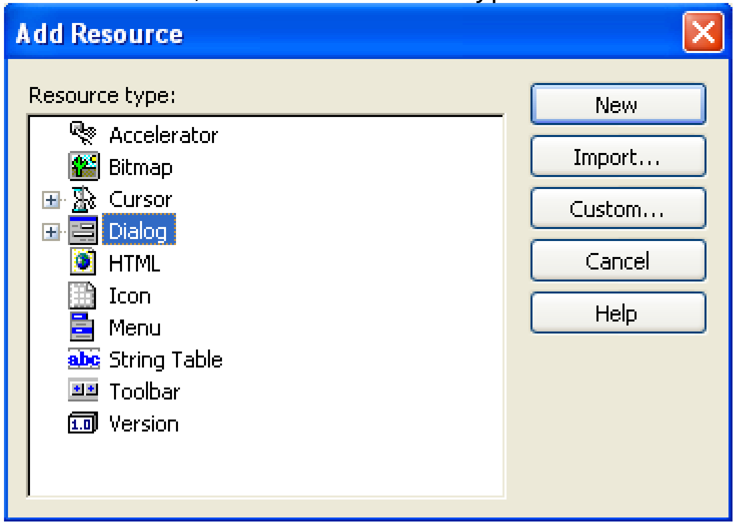


Application type:
( ) Windows application
( ) Console application
(●) DLL
( ) Static library

Additional options:
☐ Empty project
☐ Export symbols
☑ Precompiled header

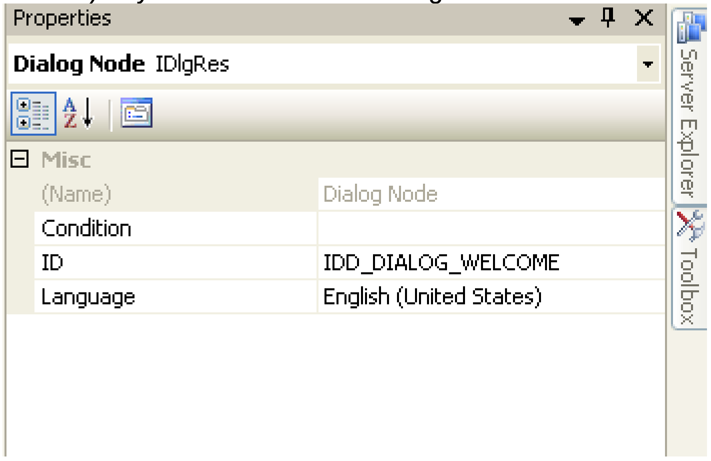
Add common header files for:
☐ ATL
☐ MFC

5. Switch to the Resource View of the project, right click on the project name to add resources, choose a resource type and click New.



6. Remember to set the Language property of the resource according to the environment in which your software is planning to install; say English(United

States) if your software is in English.

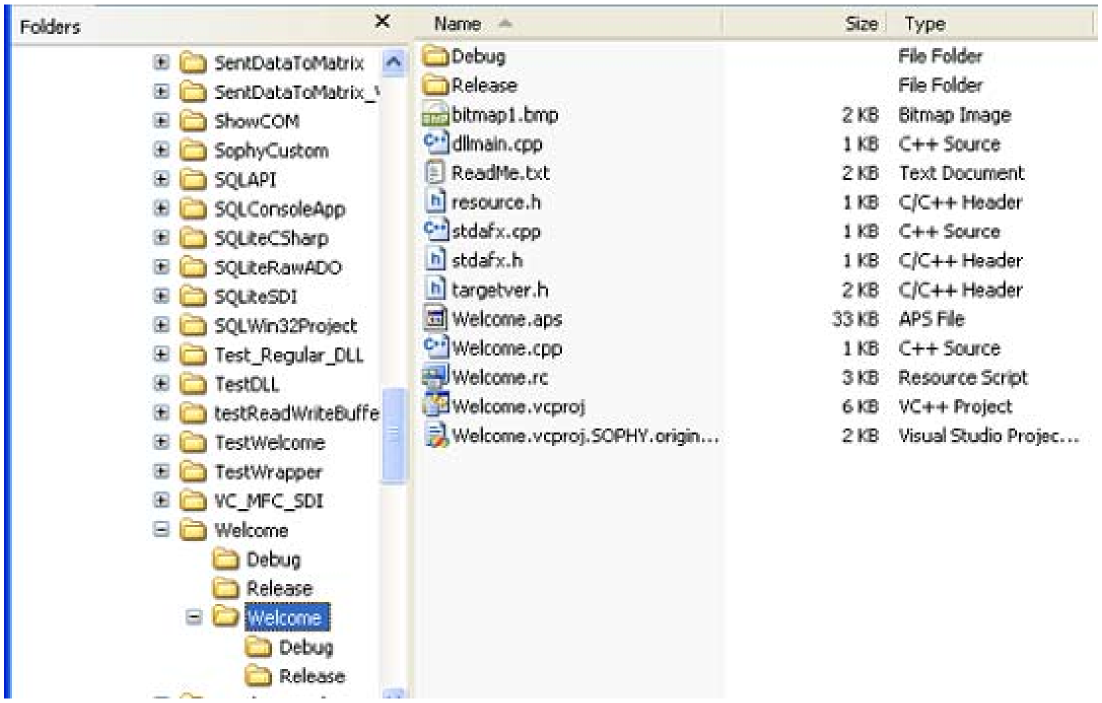


Dialog Node IDlgRes

Misc

(Name)	Dialog Node
Condition	
ID	IDD_DIALOG_WELCOME
Language	English (United States)

7. Add more controls as you want, configure the project as Debug or Release, and save the project. Then select Build>Build Solution or Rebuild Solution to build the project. You can now find a folder named "Debug" or "Release" generated under the solution folder, which contains the DLL. Files generated in this solution are as follows:



Name	Size	Type
Debug		File Folder
Release		File Folder
bitmap1.bmp	2 KB	Bitmap Image
dllmain.cpp	1 KB	C++ Source
ReadMe.txt	2 KB	Text Document
resource.h	1 KB	C/C++ Header
stdafx.cpp	1 KB	C++ Source
stdafx.h	1 KB	C/C++ Header
targetver.h	2 KB	C/C++ Header
Welcome.aps	33 KB	APS File
Welcome.cpp	1 KB	C++ Source
Welcome.rc	3 KB	Resource Script
Welcome.vcproj	6 KB	VC++ Project
Welcome.vcproj.SOPHY.origin...	2 KB	Visual Studio Projec...

16.1.2Use Resource DLL in Origin C

This section describes how to use the Resource-only DLL created in the section above.

1. Copy the DLL file to outside the Debug or Release folder, to keep the path of the DLL file the same as that of the resource.h file.
2. Start Origin and open Code Builder.
3. Create a new Origin C file named testODialog.c under the path of the DLL file. Add it to the current Workspace, and write testing code like the following. Run the OpenDlg function to open the dialog.

```
#include <Dialog.h>

#include <..\Originlab\Resource.h> //ODialog resource header



class MyDialog : public Dialog

{

public:

        // Construct dialog with dialog ID and DLL name.

        // "ODialog" is the DLL file name.

        // Not specify path, means the DLL file under the same
path of this

        // Origin C file.

        // If the DLL located at other place, should provide the
full path

        // of the DLL file.

        MyDialog() : Dialog(IDD_OC_DIALOG, "ODialog")

        {

        }



};



void OpenDlg()

{

        MyDialog odlg;

        odlg.DoModal();

}
```

16.2 Wizard Dialog

This section describes how to open a wizard dialog in Origin C. The examples in this section will use an existing wizard dialog resource DLL that gets installed with Origin C's Developer Kit. The DLL can be found in the *Samples\DeveloperKit\Dialog Builder\Wizard* sub-folder.
To open a wizard dialog we need to first define some user-defined classes. We will need a class derived from the **Dialog** class, another derived from the **WizardSheet** class, and a class for each page derived from the **PropertyPage** class.
The **WizardSheet::AddPathControl** method is used to provide a wizard map which helps the user navigate through steps or pages of a wizard. The map also allows the user to skip to any page in the wizard by clicking on the map.
The first class we define is derived from the **PropertyPage** class. This first class will contain all the information shared by all the pages in the wizard.

```
class WizPage : public PropertyPage
{
protected:
    WizardSheet* m_Sheet;
};
```

Now that we have defined our class based on **PropertyPage** we can define a class for handling each page in the wizard. These next classes will be derived from our page class defined above.

```
class WizPage1 : public WizPage
{
};

class WizPage2 : public WizPage
{
};

class WizPage3 : public WizPage
{
};
```

The next class to be defined is the place holder class. This class is derived from the **WizardSheet** class which in turn is derived from the **PropertySheet** class. This class will hold the instances of all our pages as data members.

```
class WizSheet : public WizardSheet
{
public:
    // Data members of PropertySheet are WizPage objects
    WizPage1 m_WizPage1;
    WizPage2 m_WizPage2;
    WizPage3 m_WizPage3;
};
```

With the definitions of all the pages and sheet classes completed we can now define our dialog class.

```
class WizPageDialog : public Dialog
{
```

```
public:
    // Constructor for main Dialog
    WizPageDialog(int ID) : Dialog(ID, "Wizard.DLL")
    {
    }

    // Data member of main Dialog is PropertySheet (place holder)
    WizSheet m_Sheet;
};
```

16.3 Graph Preview Dialog

This section shows how to create custom dialog with a graph preview.

16.3.1 Prepare Dialog Resource

We first need a dialog resource containing a static control, in which the preview graph will nest. Here we will use a built-in resource, IDD_SAMPLE_SPLITTER_DLG, in *OriginC\Originlab\ODlg8.dll*.

16.3.2 Prepare Source File

In Code Builder, click New button, type file name, and set Location as the same path of the above dialog resource dll oDlg8.dll - Origin install path *OriginC\Originlab* subfolder.

16.3.3 Including Needed Headers

```
//These headers contain declarations of dialog and controls
#include <..\Originlab\DialogEx.h>
#include <..\Originlab\GraphPageControl.h>
```

16.3.4 Adding User Defined Preview Class

```
//forbid some action on preview graph
#define PREVIEW_NOCLICK_BITS
(NOCLICK_DATA_PLOT|NOCLICK_LAYER|NOCLICK_LAYERICON)

#define PREVIEW_TEMPLATE    "Origin" //preview graph template

class   MyPreviewCtrl
{
public:
    MyPreviewCtrl(){}
    ~MyPreviewCtrl()
    {
        //destroy temporary books when dialog closed.
        if ( m_wksPreview.IsValid() )
            m_wksPreview.Destroy();
    }

    void    Init(int nCtrlID, WndContainer& wndParent)
    {
        //create preview graph control
        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); //first layer

        if ( !m_wksPreview )
        {
            //temporary worksheet to hold preview data.
            m_wksPreview.Create("Origin", CREATE_TEMP);
            m_wksPreview.SetSize(-1, 2); //two columns

            //long name will be displayed as axis title
            Column colX(m_wksPreview, 0);
            colX.SetLongName("Preview X");
            Column colY(m_wksPreview, 1);
            colY.SetLongName("Preview Y");

            //prepare datarange
            DataRange drPrev;
            drPrev.Add(m_wksPreview, 0, "X");
            drPrev.Add(m_wksPreview, 1, "Y");

            //plot preview curve, although it has no points now.
            int nPlot = m_glPreview.AddPlot(drPrev, IDM_PLOT_LINE);
            DataPlot dp = m_glPreview.DataPlots(nPlot);
            if ( dp ) //set preview curve color
                dp.SetColor(SYSCOLOR_RED);
        }
    }

    //update preview curve with external data.
    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; //no columns for preview

            //update source data will also update preview graph.
            dsX = vX;
            dsY = vY;
            //rescale graph for better view.
            m_glPreview.Rescale();
        }
    }
```

```
private:
    //preview graph on dialog
    GraphLayer  m_glPreview;

    //temporary worksheet to put preview data.
    Worksheet   m_wksPreview;
};
```

16.3.5 Adding Dialog Class

```
class   MyGraphPreviewDlg : public MultiPaneDlg
{
public:
    //dialog resource ID and the DLL containing it.
    MyGraphPreviewDlg() : MultiPaneDlg(IDD_SAMPLE_SPLITTER_DLG,
        GetAppPath(TRUE) + "OriginC\\Originlab\\ODlg8")
    {
    }

    ~MyGraphPreviewDlg()
    {
    }

    int     DoModalEx(HWND hParent = NULL)
    {
        InitMsgMap();

        //show dialog until user closes it.
        return DoModal(hParent, DLG_NO_DEFAULT_REPOSITION);
    }

    //message handler of dialog events
    BOOL    OnInitDialog();
    BOOL    OnDraw(Control ctrl);

protected:
    DECLARE_MESSAGE_MAP

private:
    //member stands for the preview control
    MyPreviewCtrl       m_Preview;
};
```

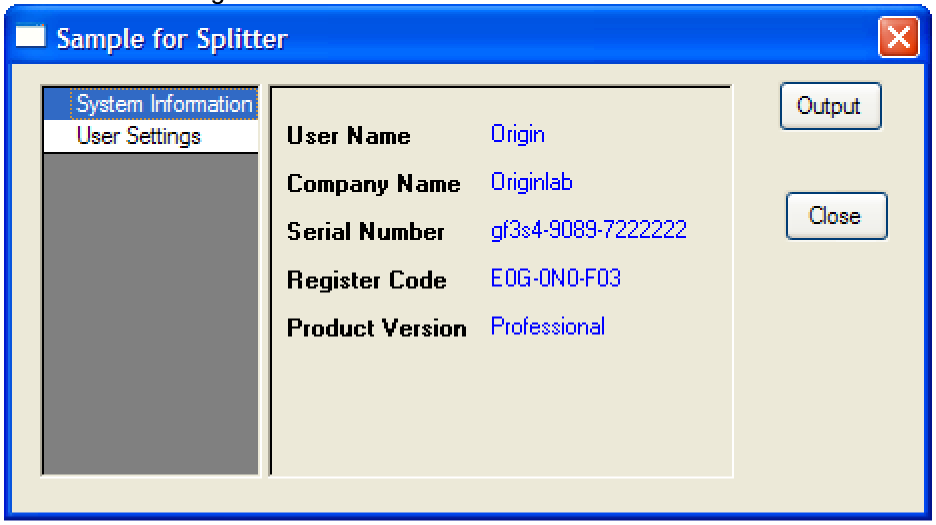
16.3.6 Open the Dialog

```
void open_preview_dlg()
{
        MyGraphPreviewDlg dlg;
        dlg.DoModalEx(GetWindow());
        return;
}
```

Execute the function above, and click the **Draw** button. You will notice the preview be updated.

16.4 Splitter Dialog

This example shows how to create a splitter dialog, which provides a better display of tree view or grid view.



User Name Origin
Company Name Originlab
Serial Number gf3s4-9089-7222222
Register Code E0G-0N0-F03
Product Version Professional

16.4.1 Prepare Dialog Resource

To create this dialog, you first must prepare a dialog resource with a Static control and two Button controls. Here we just use the existing resource IDD_SAMPLE_SPLITTER_DLG in the built-in *OriginC\Originlab\ODlg8.dll* file to simplify this example.

16.4.2 Prepare Source File

In Code Builder, click New button, type file name, and set Location as the same path of the above dialog resource dll oDlg8.dll - Origin install path *OriginC\Originlab* subfolder.

16.4.3 Including Header Files

The following header files will be used in the example. Copy the following to the above created source file.

```
#include <..\Originlab\DialogEx.h>
#include <..\Originlab\SplitterControl.h>
#include <..\Originlab\DynaSplitter.h>
```

16.4.4 Adding User Defined Splitter Class

We can derive a class from **TreeDynaSplitter**. Most dialog initialization and other event functions' code are done in a base class and make our splitter class a light class.

```
class MySplitter : public TreeDynaSplitter
{
public:
        MySplitter(){}
        ~MySplitter(){}
        //init the splitter control
        int     Init(int nCtrlID, WndContainer& wndParent, LPCSTR 
lpcszDlgName = NULL)
        {
                TreeDynaSplitter::Init(nCtrlID, wndParent, 0, 
lpcszDlgName);
                return 0;
        }
        //output current settings
        void    Output()
        {
                out_tree(m_trSettings);
        }
protected:
        // Declare message map table and message handler
        DECLARE_MESSAGE_MAP
        BOOL    OnInitSplitter();
        BOOL    InitSettings();
        void    OnRowChange(Control ctrl);

private:
        BOOL    constructSettings();
        BOOL    initSystemInfo(TreeNode& trSys);//show system 
information
        BOOL    initUserInfo(TreeNode& trUser);//to collect user 
settings.

private:
        GridTreeControl m_List; //grid control on left panel
        Tree    m_trSettings;//splitter tree on right panel
        bool    m_bIsInit;//indicate whether it is from init event

};

//map the control messages and events.
BEGIN_MESSAGE_MAP_DERIV(MySplitter, TreeDynaSplitter)

        ON_INIT(OnInitSplitter) //init splitter settings
        //save splitter size & position when destroy
        //this is done in base class.
        ON_DESTROY(OnDestroy)
        ON_SIZE(OnCtrlResize)
        //when control is ready, need to resize the splitter and its 
position
        ON_USER_MSG(WM_USER_RESIZE_CONTROLS, OnInitPaneSizs)

        //when user select different row on left panel
        ON_GRID_ROW_COL_CHANGE(GetMainPaneID(), OnRowChange)
END_MESSAGE_MAP_DERIV

BOOL    MySplitter::OnInitSplitter()
{
        TreeDynaSplitter::OnInitSplitter(&m_List);
        constructSettings();   //construct tree settings
        InitSettings(); //tree settings to splitter GUI
        SetReady();
        return TRUE;
}
```

```
//when user selects a different row, update right panel
void    MySplitter::OnRowChange(Control ctrl)
{
        if ( !m_bReady )
                return;
        //show sub nodes under current branch
        TreeNode trCurrent = ShowListContent(-1, true, m_bIsInit);
        if ( trCurrent )
        {
                //load settings from registry
                string strTag = trCurrent.tagName;
                LoadBranchSetting(GetDlgName(), strTag);
        }
        m_bIsInit = false;
        return;
}

//init splitter settings
BOOL    MySplitter::InitSettings()
{
        m_bIsInit = true;
        ///set not ready, avoid flash and painting problem on GUI
        m_bReady = false;
        //set the splitter tree for display
        ShowList(m_trSettings, ATRN_STOP_LEVEL);
        m_bReady = true; //reset ready state.
        SelectRow(0); //select first row.
        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;
}

//display your Origin's basic information.
//you can also display OS related information here.
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;
}

//controls to collect user information and settings.
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;
}
```

16.4.5 Adding User Defined Splitter Dialog Class

The splitter dialog contains a splitter control object, so the dialog can initialize the splitter control and post messages to it on the proper events.

```
//dialog name, which will also be used to save settings in registry
#define STR_DLG_NAME    "My Splitter Dialog"
class MySplitterDlg : public MultiPaneDlg
{
```

```
public:
        //resource ID and which DLL contains this dialog resource
        MySplitterDlg() : MultiPaneDlg(IDD_SAMPLE_SPLITTER_DLG, "ODlg8")
        {
        }
        ~MySplitterDlg()
        {
        }
        //open dialog until user close it.
        int     DoModalEx(HWND hParent = NULL)
        {
                //set up message map
                InitMsgMap();
                return DoModal(hParent, DLG_NO_DEFAULT_REPOSITION);
        }
        //init controls and other settings before dialog open
        BOOL    OnInitDialog();
        //when dialog initialization finish
        BOOL    OnReady();
        //when user click 'Output' button
        BOOL    OnOutput(Control ctrl);
protected:
        DECLARE_MESSAGE_MAP
private:
        MySplitter     m_Splitter;

};

//map dialog message
BEGIN_MESSAGE_MAP(MySplitterDlg)
        ON_INIT(OnInitDialog)
        ON_READY(OnReady)
        ON_BN_CLICKED(IDC_LOAD, OnOutput)
END_MESSAGE_MAP

BOOL    MySplitterDlg::OnInitDialog()
{
        //rename buttons title to meaningful text
        GetDlgItem(IDC_LOAD).Text = "Output";
        GetDlgItem(IDCANCEL).Text = "Close";
        m_Splitter.Init(IDC_FB_BOX, *this, STR_DLG_NAME);
        return TRUE;
}

BOOL    MySplitterDlg::OnReady()
{
        //update dialog
        UpdateDlgShow();
        SetInitReady();
        //set splittercontrol ready as to init the position and size
        m_Splitter.OnReady();
        return TRUE;
}

BOOL    MySplitterDlg::OnOutput(Control ctrl)
{
        //dump current user settings.
        m_Splitter.Output();
        return TRUE;
}
```

16.4.6 Open Dialog

After the steps above, save all the code and build it, then execute the following function to open the splitter dialog.

```
void    test_MySplitterDlg()
{
        MySplitterDlg dlg;
        dlg.DoModalEx(GetWindow());
}
```

17 Accessing External Resources

Origin C can access external DLLs and, in addition, applications outside of Origin can be added using automation (COM) server capability.

17.1 Access an External DLL

Origin C can make calls to functions in external DLLs created by C, C++, C++(.Net), C# or Fortran compilers. To do this, you need to provide the prototype of a function in a header file and tell Origin C which DLL file contains the function body. Assume the functions are declared in a header file named **myFunc.h**. You should **include** this file in your Origin C file where you want to call those functions, like:

```
#include <myFunc.h> //in the \OriginC\System folder
#include "myFunc.h" //in the same folder as your Origin C code
#include "C:\myFile.h" //in specified path
```

Then you should tell Origin C where to link the function body, and you must include the following Origin C **pragma** directive in the header file **myFunc.h**, just before your external DLL function declarations. Assume your DLL file is **UserFunc.dll**:

```
#pragma dll(UserFunc) //in the same folder as your Origin C code
#pragma dll(C:\UserFunc) //in specified path
#pragma dll(UserFunc, header) //in the same folder as this .h file
#pragma dll(UserFunc, system) //in the Windows system folder
```

The Origin C compiler supports three calling conventions: **__cdecl**(default), **__stdcall** and **__fastcall**. These calling conventions determine the order in which arguments are passed to the stack as well as whether the calling function or the called external function cleans the arguments from the stack.
Notes: you don't need to include the .dll extension in the file name. And all function declarations after the **pragma** directive will be considered external and from the specified DLL. This assumption is made until a second **#pragma dll(filename)** directive appears, or the end of the file is reached.
A good and complete example of how to access an external DLL is Accessing SQLite Database. There are other Origin sample projects demonstrating how to call a function from a C dll or a Fortran dll in Origin C. They can be found in the *\Samples\Origin C Examples\Programming Guide\Calling Fortran* and *\Samples\Origin C Examples\Programming Guide\Calling C DLL* subfolders of Origin.

17.1.1Calling GNU Scientific Library

This article demonstrate how to use GSL in Origin C. First you need the GSL dll, you can check here to see how to build GSL dlls, or you can just download them(dlls) from http://gnuwin32.sourceforge.net/packages/gsl.htm. You need just two dlls (libgsl.dll and libgslcblas.dll), and you can put them into the same folder where you are going to keep your Origin C files. For example, in a folder called c:\oc\.

libgsl.dll

This is the main gsl dll

libgslcblas.dll

This dll is needed by libgsl.dll

To use libgsl.dll in Origin C, you will need a header file that provides the prototypes of the gsl functions. You can copy and translate(if needed) the necessary prototype/definition from GSL header files, for example, call it **ocgsl.h**, and create in the c:\oc\ folder.

ocgsl.h

```
#pragma dll(libgsl, header)
// this is OC special pragma,
// header keyword is to indicate libgsl.dll is in same location as this
file

#define GSL_EXPORT     // for OC, this is not needed, so make it empty

// you can directly search and copy gsl function prototypes here

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);
```

The following is a simple OC file to show how to call gsl_sf_zeta_int and gsl_fit_linear

test_gsl.c

```
#include <Origin.h>
#include "ocgsl.h"


// Example of using Riemann Zeta Function in GSL
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);
}


// Example of using linear fit in GSL
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);
}
```

Using GSL in a Fitting Function

There is also an exampleto show how to use gsl functions in a fitting function.

Notice on using GSL functions

Origin C doesn't support external functions that return a struct type variable, so those functions return this kind of data can not be used in Origin C, e.g.

```
gsl_complex gsl_complex_add (gsl_complex a, gsl_complex b)
```

since it returns a gsl_complex type of data, and gsl_complex is defined as:

```
typedef struct
{
    double dat[2];
}gsl_complex;
```

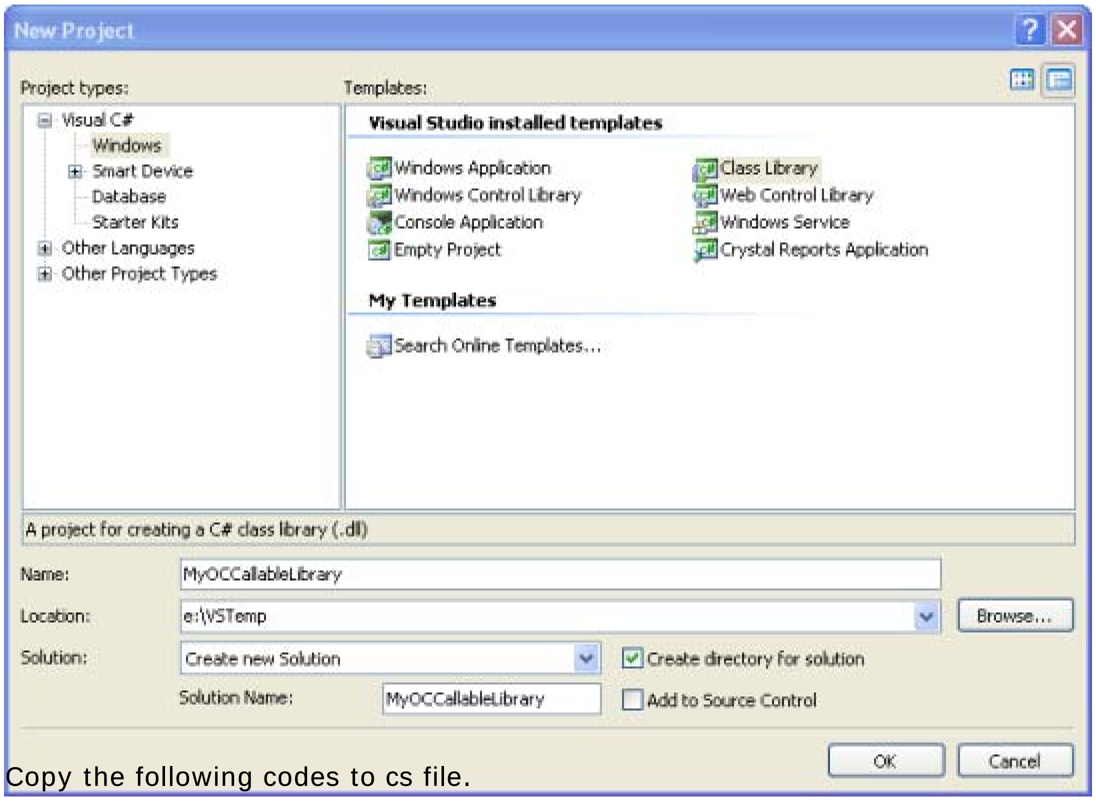
17.1.2 Access CPlusPlus(.Net) And CSharp DLL

This charpter will introduce how to access the DLL created by C++(.Net) or C# in Origin C.

Access C# Class DLL

The following example shows how to create a DLL in Microsoft Visual Studio 2005 with C#, and then access it in Origin C. This Dll privode a class with properties and functions. Function Sum shows how to pass data array from Origin C vector to C# function.

1. In Microsoft Visual Studio 2005, select File -> New -> Project..., in New Project dialog, choose the settings as below:



A project for creating a C# class library (.dll)

Name: MyOCCallableLibrary

Location: e:\VSTemp

Solution: Create new Solution

Solution Name: MyOCCallableLibrary

☑ Create directory for solution

☐ Add to Source Control

2. Copy the following codes to cs file.

```
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;

        }

    }

}
```

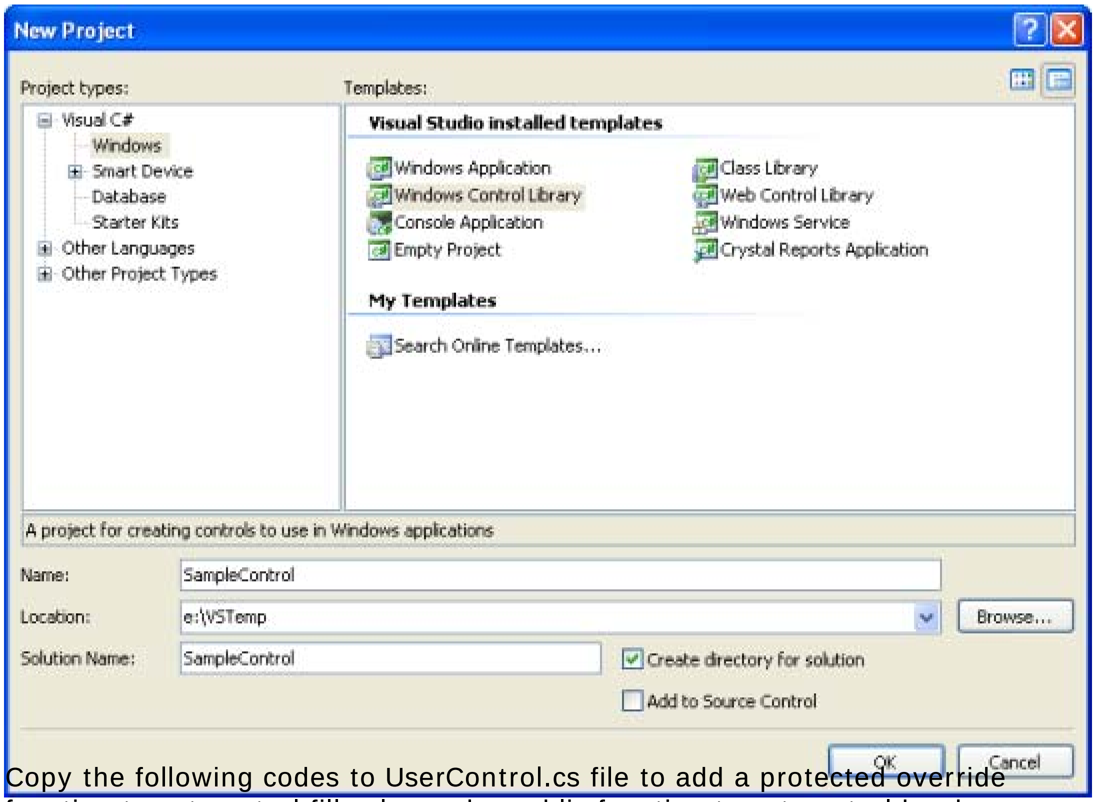
3. Choose menu Tools -> Create GUID, in opening dialog select Registry Format radio button, click New GUID button and then click Copy button. Paste this GUID to replace that one in [Guid("...")] in the above codes. Reproduce this action again to replace the second GUID in code.
4. Press F6 to build solution.
5. Start Origin, open Code Builder, new a c file, then copy the following Origin C code to c file.

```
void access_DLL()

{

        Object obj = CreateObject("temp.NET_Temperature");


        obj.Celsius = 0; // access property

        out_double("", obj.GetFahrenheit()); // access function


        obj.Fahrenheit = 300;

        out_double("", obj.GetCelsius());


        vector vec;

        vec.Data(1,10,1);

        _VARIANT var = vec.GetAs1DArray();

        out_double("", obj.Sum(var));

}
```

Access C# and C++ Resource DLL

The following shows how to create an ActiveX control by C#, and use it in a dialog in Origin C.
Steps 1~7 shows how to create an C# ActiveX control in Microsoft Visual Studio 2005.

1. Start Microsoft Visual Studio 2005, choose File -> New -> Project... to open New Project dialog, do the following settings, then click OK button.



A project for creating controls to use in Windows applications

Name: SampleControl

Location: e:\VSTemp

Solution Name: SampleControl

☑ Create directory for solution

☐ Add to Source Control

2. Copy the following codes to UserControl.cs file to add a protected override function to set control fill color and a public function to set control border.

```
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();

        }

    }


    //declare an Interface for the control settings

    [Guid("CCBD6133-813D-4dbb-BB91-16E3EFAE66B0")]

    public interface UserControl1Interface

    {

        void SetBorder(bool bSet);

    }

}
```

3. Select Tools -> Create GUID to open Create GUID dialog, choose Registry Format radio button, click New GUID and Copy button to copy the newly created GUID. Use the new GUID to replace the one in above code [Guid("...")].
4. Choose UserControl1.cs[Design] tab, in Properties window, click Events button ⚡, drag scroll bar to choose MouseClick and double click it to add mouse click event function into UserControl.cs file. Use the same method to add mouse double click event.
5. In UserControl1l.cs file, copy the following code outside UserControl1 class.

```
//declare delegates for events

public delegate void MouseAction(int x, int y);
```

6. In UserControl1 class, add the following implements.

```
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. Outside UserControl1 class, add the following interface.

```
//declare interface for events

[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. According to Dialog Builder: Simple Hello World Dialog: Create Resource-only DLL in Visual Studio 2008 chapter to generate dialog resource DLL in Microsoft Visual Studio 2008.

9. Use ActiveX control in dialog and show dialog by Origin C. In Origin Code Builder, new a c file, and copy the following code to it.

```
//Dialog and control Ids. Need change dialog id according to real
case.

#define IDD_DIALOG1                     101

#define IDC_DOTNET                                          1000



//Exposed events from Control

#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)
        //Event handler entries for the exposed events

        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;

};



// not specify DLL file path here, just assume the dll file under
```

```
the same

// path with the current c file.

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))

        {

                //Control intialization using the exposed
interface of

                //the DotNet control

                Object ctrlObj = m_ctrlDotNet.GetActiveXControl();

                ctrlObj.SetBorder(true);

        }
        return TRUE;

}



void Launch_CDotNetComInteropDlg()

{

        CDotNetComInteropDlg dlgDotNet;

        dlgDotNet.DoModal();

}
```

17.2 Access an External Application

The Microsoft Component Object Model (COM) is a software architecture that allows applications to be built from a binary software component, and it makes the development of software much easier and more effective.
Origin provides the capability for COM client programming, and it is supported only in **OriginPro**. This mechanism uses **Object** type to represent all COM (automation server) objects. All COM objects can be initialized in two ways:

```
//by the CreateObject method
Object oExcel;
oExcel = CreateObject("Excel.Application");

//by initialization from an existing COM object.
Object oWorkBooks;
oWorkBooks = oExcel.Workbooks;
```

Origin C also provides a class to access **Matlab**, which enables communication between Origin and Matlab.

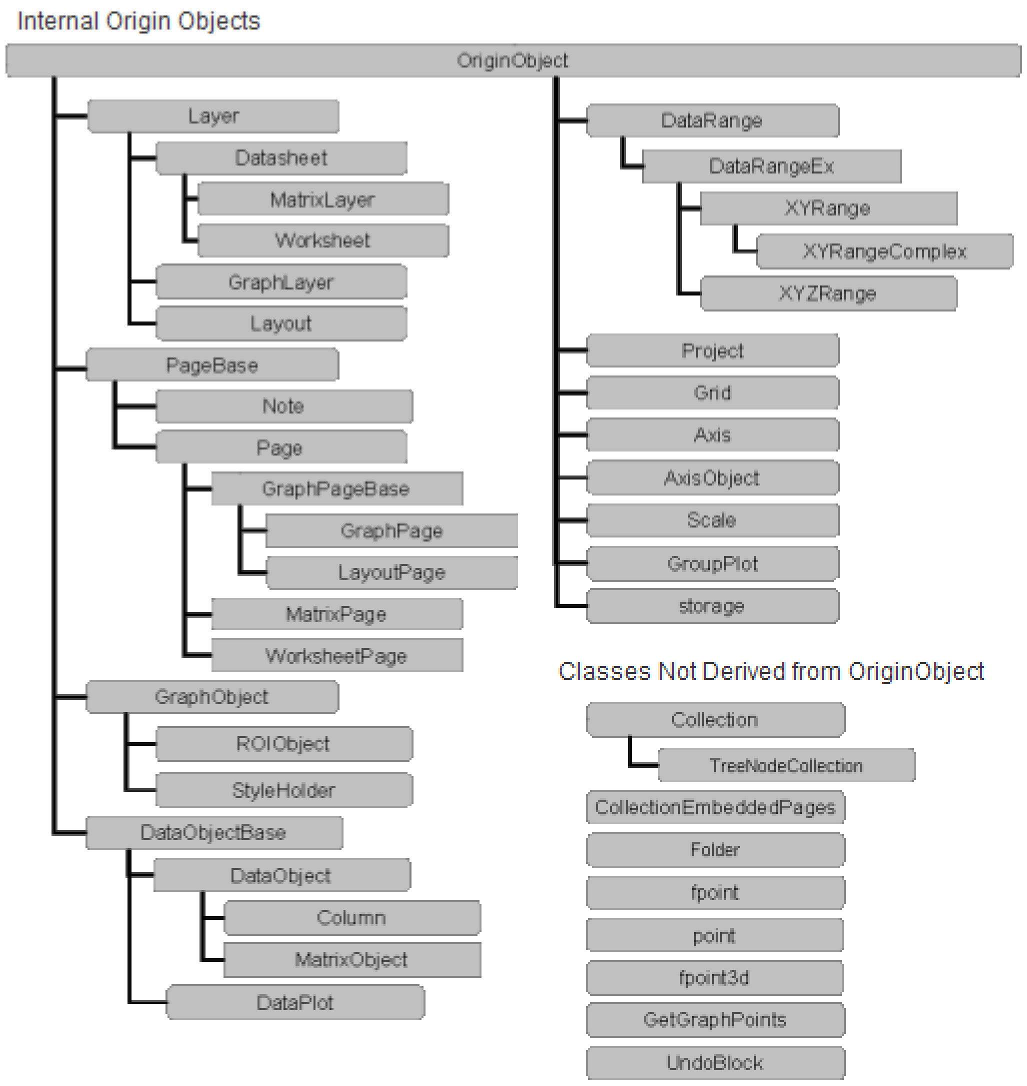
```
#include <Origin.h>
#include <externApps.h> //required header for the MATLAB class
void test_Matlab()
{
    Matlab matlabObj(true);
    if(!matlabObj)
    {
        out_str("No MATLAB found");
        return;
    }

    //defines 3x5 matrix named ma
    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);// show str from MATLAB

    // put matrix into Origin matrix
    MatrixLayer matLayer;
    matLayer.Create();
    Matrix mao(matLayer);

    //Transfer MATLAB's matrix (ma) to Origin's mao matrix.
    BOOL bRet = matlabObj.GetMatrix("ma", &mao);
}
```

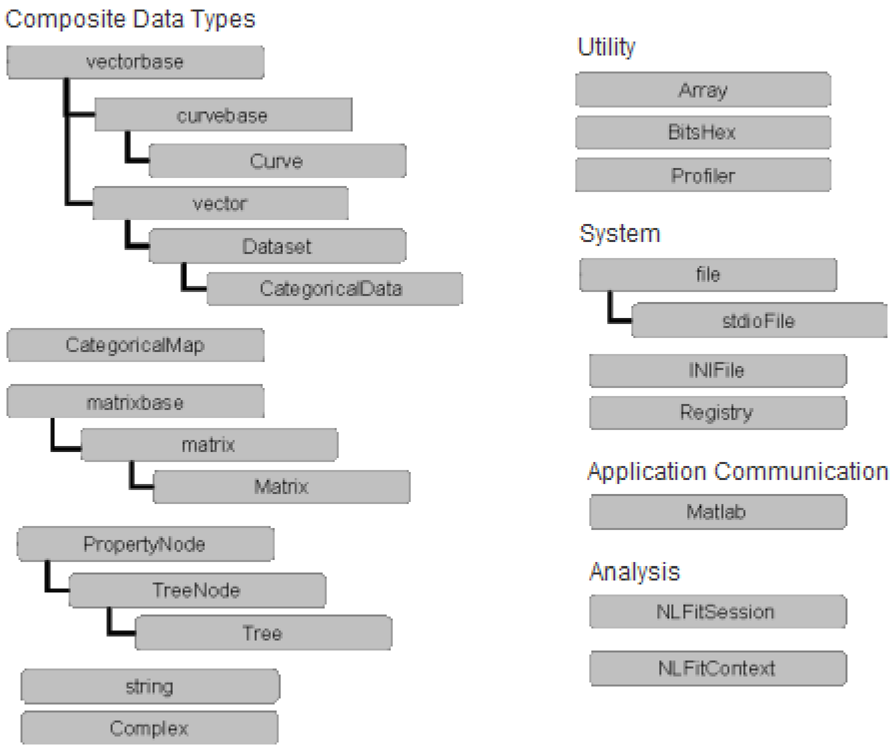
COM client programming in Origin C can be used to programmatically exchange data with MS Office applications. There is a comprehensive example demonstrating how to read data from Excel worksheets, plot a graph in Origin, and place it in a Word document. This example can be found in the *\Samples\COM Client\MS Office* subfolder in Origin.
(Origin C COM programming can also be used to communicate with databases by accessing an ActiveX Data Object (ADO), and there is a sample file demonstrating that SQL and Access databases can be imported to a worksheet, and subsequent data modifications returned to the database. For more information, see the file ADOSample.c in the *Samples\COM Clients\ADO* subfolder of Origin.

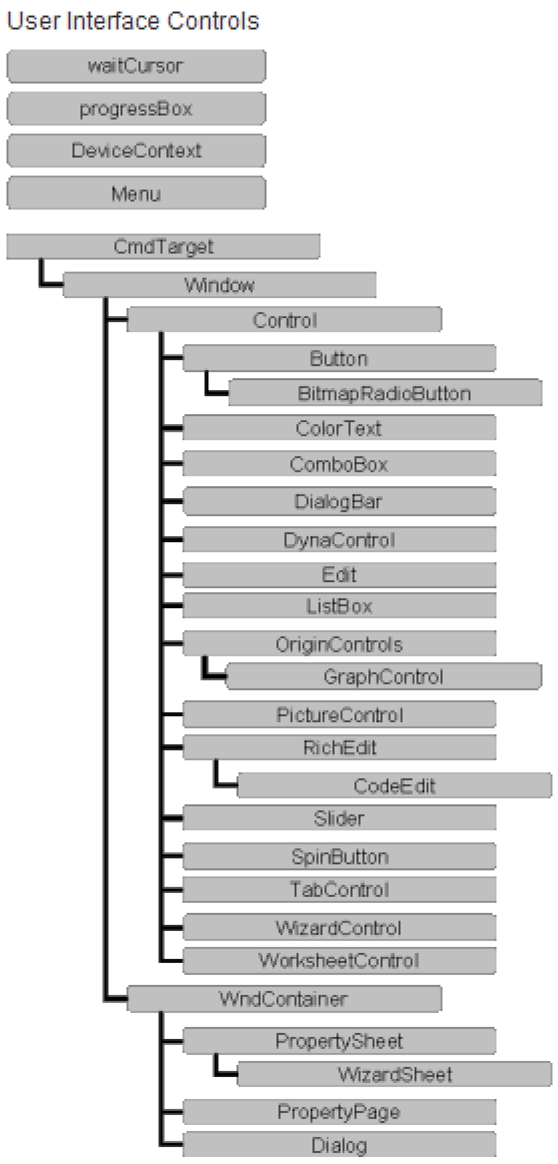
18 Reference

18.1 Class Hierarchy

The following schematic diagram shows the hierarchy of built-in Origin C classes.




Composite Data Types
vectorbase
curvebase
Curve
vector
Dataset
CategoricalData
CategoricalMap
matrixbase
matrix
Matrix
PropertyNode
TreeNode
Tree
string
Complex
Utility
Array
BitsHex
Profiler
System
file
stdioFile
INIFile
Registry
Application Communication
Matlab
Analysis
NLFitSession
NLFitContext



18.2 Collections

The Collection class provides a template for holding multiple objects of the same type. In Origin C, an instance of a Collection is read-only. You cannot add items to, or delete items from, a collection.
There are a number of Origin C classes that contain data members which are instances of a Collection. For example, the Project class contains a data member named WorksheetPages. The WorksheetPages data member is a collection of all the existing WorksheetPage objects in the project.
The table below lists each of the Origin C Collections, along with the class that contains the collection and the item types in the collection.

Class	Data Member	Collection of
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	Pages	PageBase
Project	WorksheetPages	WorksheetPage
Selection	Objects	OriginObject
TreeNode	Children	TreeNode
Worksheet	Columns	Column
Worksheet	EmbeddedPages	Page

Examples

List all graph pages in the current project.

```
foreach(GraphPage gp in Project.GraphPages)
{
	out_str(gp.GetName());
}
```

List all worksheet pages in the current project.

```
foreach(WorksheetPage wksPage in Project.WorksheetPages)
{
	out_str(wksPage.GetName());
}
```

List all matrix pages in the current project.

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

```
{
        out_str(matPage.GetName());
}
```

List all data plots.

```
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);
}
```

18.3 X-Function Option Strings

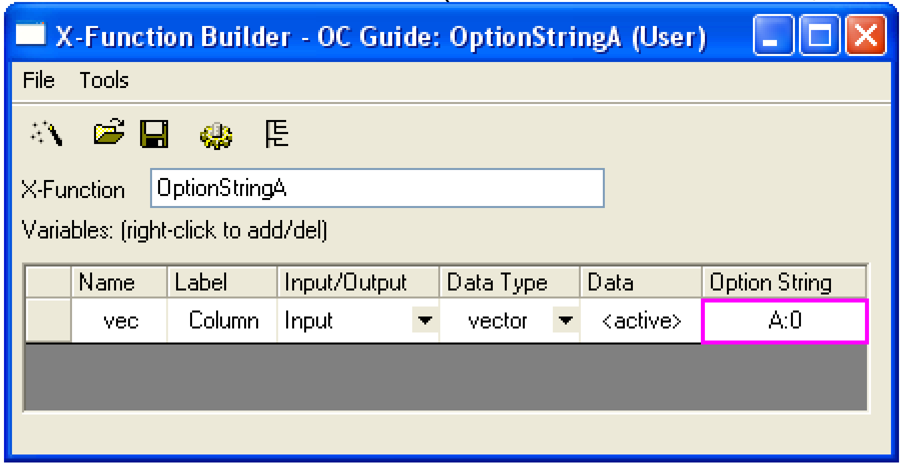
18.3.1A - Recalculate Mode

Recalculate Options enable you to control the **Recalculate** mode of an X-Function. By default, when the X-Function includes one of the following Origin Object types as an input or output variable: XYRange, XYZRange, vector, Column, matrix and MatrixObject, it allows you to use **Manual**, **Auto** or **None** as the **Recalculate** mode (the default mode is **Manual**). Please note that if the variable is used as both an input and output variable, the **Recalculate** mode is forbidden, and the **Recalculate** combo box will be hidden.

A:0x00

Hides the **Recalculate** combo box from the X-Function dialog.
Example

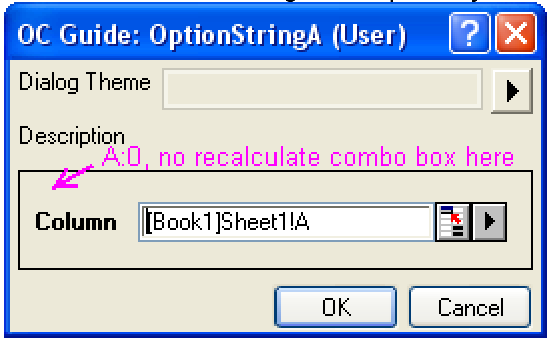
1. Create a new X-Function with the name and variables shown in the following image. Then click the **Save** button to save the X-Function in *<User Files Folder>\X-Functions\OC Guide* (if this folder does not exist, create it).



X-Function: OptionStringA
Variables: (right-click to add/del)

Name	Label	Input/Output	Data Type	Data	Option String
vec	Column	Input	vector	<active>	A:0

2. Run this X-Function with the command `OptionStringA -d;` in the Command Window. In the dialog that opens, you will not see the **Recalculate** combo box.


OC Guide: OptionStringA (User)

Dialog Theme

Description

A:0, no recalculate combo box here

Column [Book1]Sheet1!A

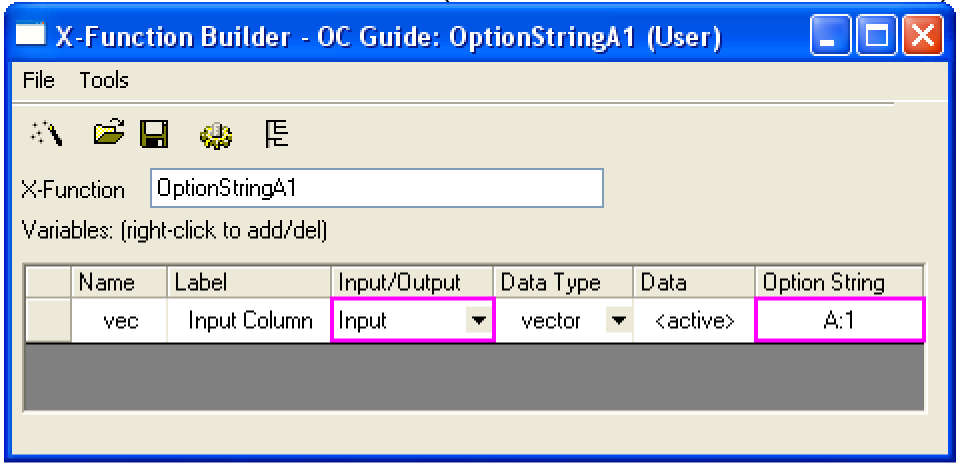
OK Cancel

A:0x01

Sets **None** as the default **Recalculate** mode. This mode is available for both input variables and output variables.

Example

1. Create a new X-Function with the name and variables shown in the following image. Then click the **Save** button to save the X-Function in *<User Files Folder>\X-Functions\OC Guide* (if this folder does not exist, create it).


X-Function Builder - OC Guide: OptionStringA1 (User)

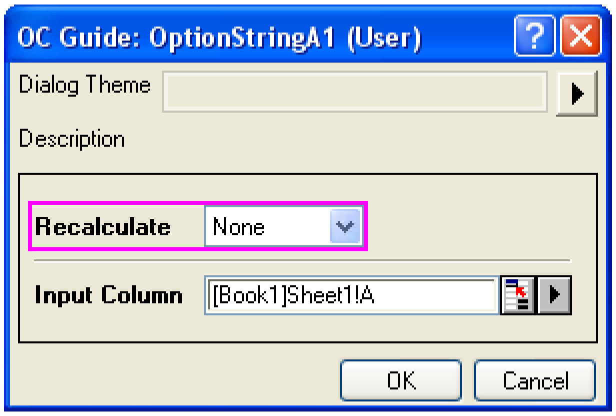
File Tools

X-Function OptionStringA1

Variables: (right-click to add/del)

Name	Label	Input/Output	Data Type	Data	Option String
vec	Input Column	Input	vector	<active>	A:1

2. Run this X-Function with the command `OptionStringA1 -d;` in the Command Window. In the dialog that opens, the **Recalculate** combo box shows with **None** selected.


OC Guide: OptionStringA1 (User)

Dialog Theme

Description

Recalculate None

Input Column [Book1]Sheet1!A

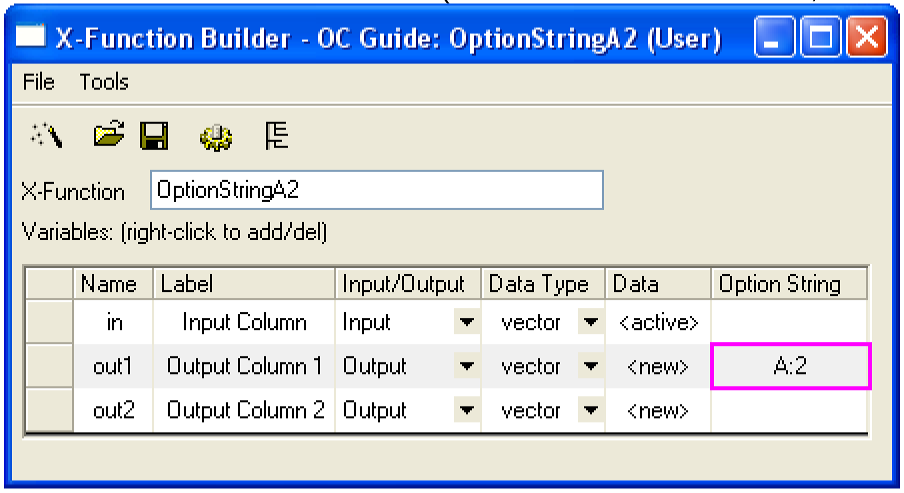
OK Cancel

A:0x02

Allows sorting on the output columns with the **Recalculate** mode. It works for output variables only.

Example

1. Create a new X-Function with the name and variables shown in the following image. Then click the **Save** button to save the X-Function in <*User Files Folder*>*\X-Functions\OC Guide* (if this folder does not exist, create it).

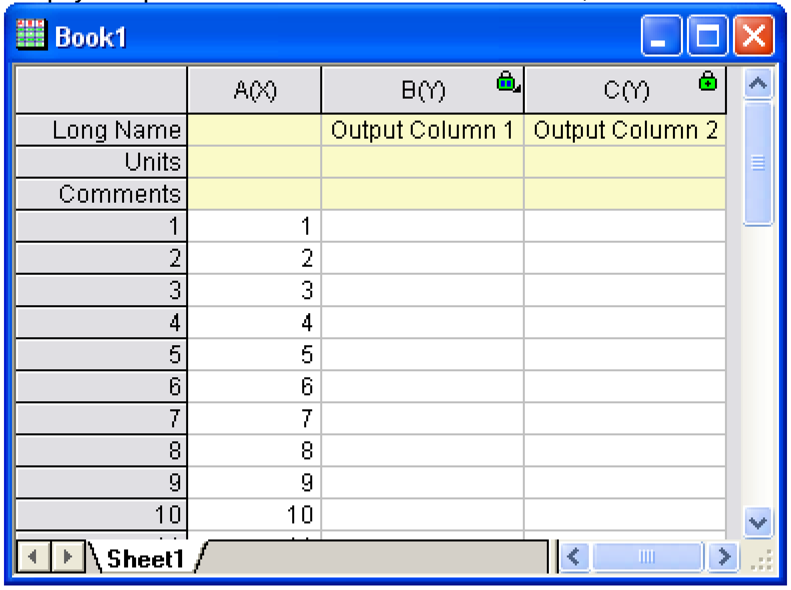


X-Function: OptionStringA2

Variables: (right-click to add/del)

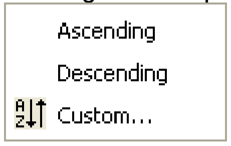
Name	Label	Input/Output	Data Type	Data	Option String
in	Input Column	Input	vector	<active>	
out1	Output Column 1	Output	vector	<new>	A:2
out2	Output Column 2	Output	vector	<new>	

2. Create a new worksheet with one column and fill this column with row numbers. Then highlight the column and run the X-Function with the command `OptionStringA2 -d;` in the Command window. After clicking the **OK** button, two empty output columns with locks will show, as below.



	A(X)	B(Y)	C(Y)
Long Name		Output Column 1	Output Column 2
Units			
Comments			
1	1		
2	2		
3	3		
4	4		
5	5		
6	6		
7	7		
8	8		
9	9		
10	10		

3. Click on the header of column B to highlight it, then move the mouse to the **Sort Worksheet** item of the right-click menu. The sub context menu is available for sorting the output columns.



A:0x04

Allows the **Recalculate** mode, but the **Recalculate** combo box will be hidden in the X-Function dialog.

18.3.2B - Browser Dialog Options

Browser Dialog Options control the settings in the **Page / Graph Browser** dialog, which is opened by a button in the X-Function dialog. They can also be used to filter the opened dialog. Please refer to the example about X-Function Graph Browser dialog.

B:0x0001

Sorts all pages in the **Page / Graph Browser** once the dialog opens.

B:0x0002

Excludes 3D graphs from the **Graph Browser**.

18.3.3C - Miscellaneous Options

C:0x0001

Disables composite ranges from assigning column designation. Take an output variable with XYRange type, for example. If this option string is not specified for the variable, there will be two output columns, one with X designation and the other with Y designation. Otherwise, the two columns will both have the default Y designation.

C:0x0002

Creates a new page with hidden state.

C:0x0004

Makes an X-Function non-undoable.

C:0x0010

By default, if a variable with double type is missing a value, this variable will show nothing (empty cell). This option string is used to show this kind of empty variable as "--".

C:0x0100

This option string is only available for output variables of Range type. If the output variable is set to <*new*>, a valid Origin C Range object will be created, but without a new column for it. This option string helps the user prepare rows and columns for the output range objects inside the X-Function body.

C:0x0200

This option string is only available for output variables of Range type. If the output variable is set to <*new*>, it will replicate the input Range, with the same number of columns and rows.

18.3.4E - Execution Control

E:V

Used in the X-Function wizard and X-Function bar to keep the real value of a variable. The destination created by the first **Apply** button will be used by the subsequent **Apply** buttons.

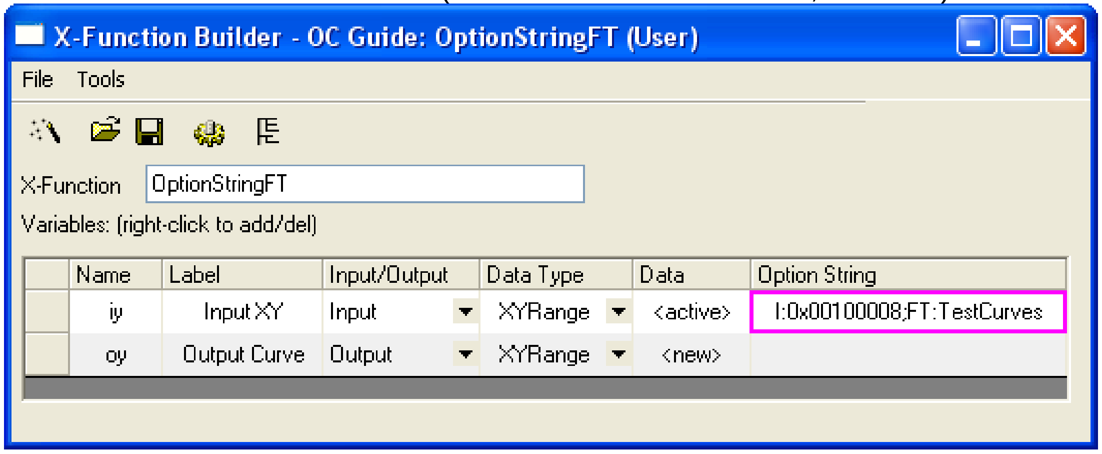
18.3.5F - Dialog Numeric Display Format

F:*6*
Provides a format string for displaying numeric values with double type. The standard LabTalk numeric format notation, such as *(Origin global numeric format setting), *.2*(two decimal digits), etc., can be used.

18.3.6FT - Data Plot Selection Filter

FT:str1|str2
Specifies the tags for the action filter of data selection from a graph. If a data object contains tags *str1* or *str2*, it will be ignored when selecting the data hunting menu item **Add all plots in active page/layer**.
Example
This example will show how to use the **FT** option string to exclude the data plot with tag name *TestCurves* when selecting all plots from a graph.

1. Create a new X-Function with the name and variables shown in the following image. Then click the **Save** button to save the X-Function in *<User Files Folder>\X-Functions\OC Guide* (if this folder does not exist, create it).



X-Function Builder - OC Guide: OptionStringFT (User)

File Tools

X-Function: OptionStringFT

Variables: (right-click to add/del)

Name	Label	Input/Output	Data Type	Data	Option String
iy	Input XY	Input	XYRange	<active>	I:0x00100008;FT:TestCurves
oy	Output Curve	Output	XYRange	<new>	

2. Open this X-Function in **Code Builder**, then copy the following code and paste it into the body of the main function, **OptionStringFT**, then click the **Compile** button Compile.

```
// Put data to output XYRange

vector vx, vy;

vx.Data(1, 10, 1);

vy = vx;
```

```
oy.SetData(&vy, &vx);



// The specified name should be the same as options string "FT"
of iy

tag_columns_in_data_range(oy, "TestCurves");



// Get source range graph, and plot results on this graph

vector<uint> vnIDs;

if( iy.GetPlots(vnIDs) > 0 )

{

        DataPlot dp;

        dp = (DataPlot)Project.GetObject(vnIDs[0]);

        GraphLayer gl;

        dp.GetParent(gl);



        gl.AddPlot(oy);

}
```

3. Import the file *<Origin Installation Folder>\Samples\Curve Fitting\Multiple Gaussians.dat* to a new worksheet. Highlight all columns and make a graph with line plots. Activate the newly created graph and run `OptionStringFT -d;` in the Command Window, then click the **OK** button to generate the output XY columns in the source worksheet and add its plot to the source graph.

4. Activate the graph and run `OptionStringFT -d;` again to open the X-Function dialog. In the **Input XY** branch, the newly outputted data plot from above is not there, because of the tag name *TestCurves* of this data plot.

18.3.7FV - Specify the Source of Output Variable

FV:varname specifies the name of the input variable for each output variable. Then the information of this variable's book/sheet/object will be traced and shared with the output variable when the output variable is set by *<input>/<same>*. For example, suppose an X-Function contains three XYRange variables: the first two are input variables and the last one is an output variable. If the name of one input variable is set to the output variable by this option string, the output columns will be the same as the specified input columns when output is set as <*input*> or <*same*>. Otherwise, the output columns will be the same as the first input columns.

18.3.8G - Grouping Controls on GUI

Used to group related variables in the X-Function dialog. Add **G:*Group Name*** in the **Option String** field of the variable for the beginning of the group, and **G** in the **Option String** field of another variable for the end of the group.

G:*Group*

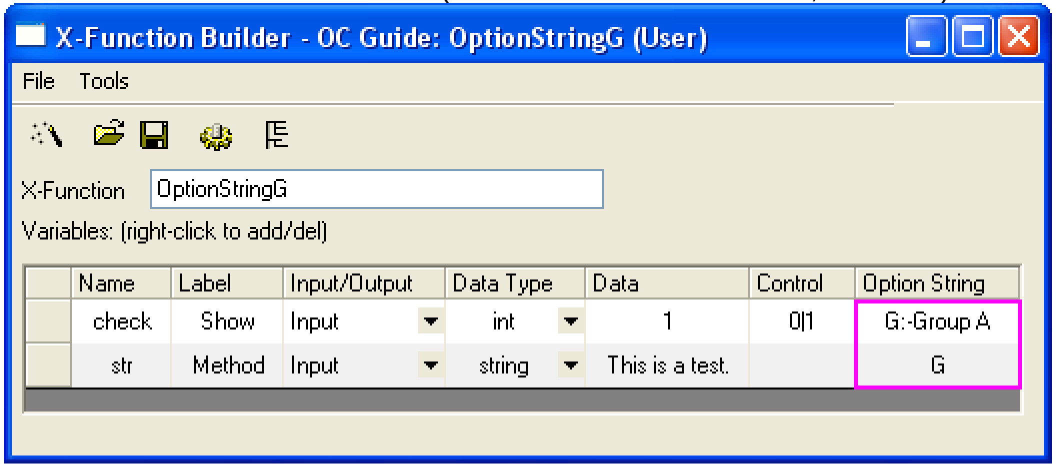
The beginning of the group. By default, the branch of the group is closed. To set the branch as open for the first time, open the dialog and add a dash before *Group Name*, like **G:-*Group Name***. Then the dialog will remember the branch status when opened next time.

G

The end of the group.

Example

1. Press F10 to open X-Function Builder. Create an X-Function as the following image shows and then click the **Save** button to save it into <*Origin User Files Folder*>*\X-Functions\OC Guide* (if this folder does not exist, create it).

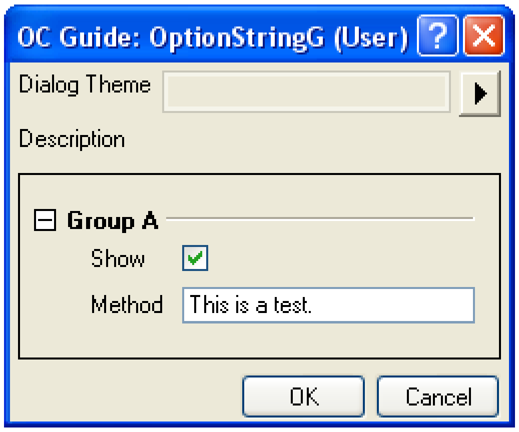


X-Function: OptionStringG

Variables: (right-click to add/del)

	Name	Label	Input/Output	Data Type	Data	Control	Option String
	check	Show	Input	int	1	0|1	G:-Group A
	str	Method	Input	string	This is a test.		G

2. Run `OptionStringG -d;` in the Command window. The dialog that opens will look like the one below.



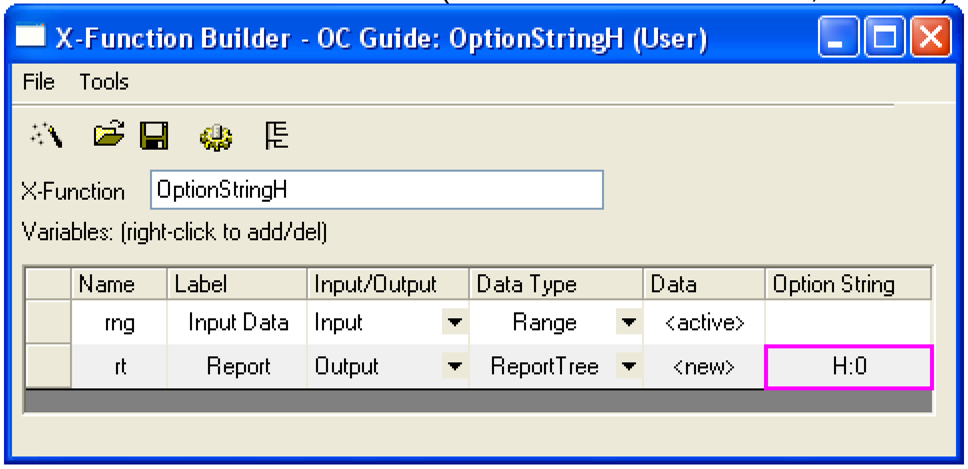
Group A

Show ✓

Method: This is a test.

18.3.9H - Setting Result Tables Shown Style

1. Create a new X-Function with the name and variables shown in the following image. Then click the **Save** button to save the X-Function in *<User Files Folder>\X-Functions\OC Guide* (if this folder does not exist, create it).



X-Function OptionStringH

Variables: (right-click to add/del)

Name	Label	Input/Output	Data Type	Data	Option String
rng	Input Data	Input	Range	<active>	
rt	Report	Output	ReportTree	<new>	H:0

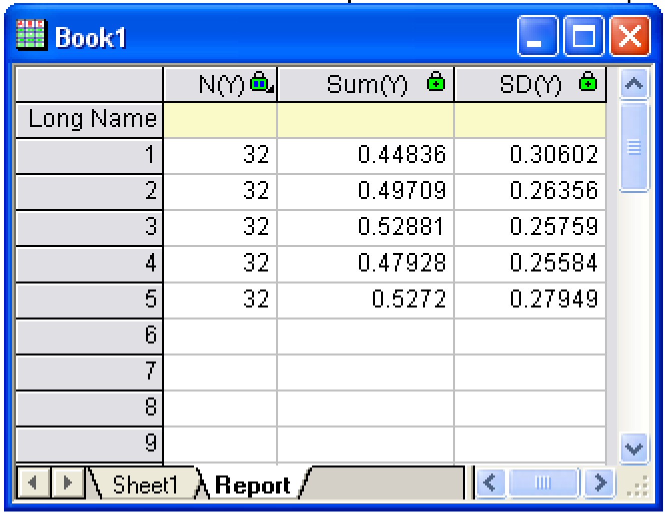
2. Open this X-Function in **Code Builder**, and add the following needed header file after the line *//put additional include files here*.

```
#include <ReportTree.h>
```

3. Copy the following code into the main function, **OptionStringH**, which is used to do basic statistical analysis on the selected data and then prepare the report sheet for the results.

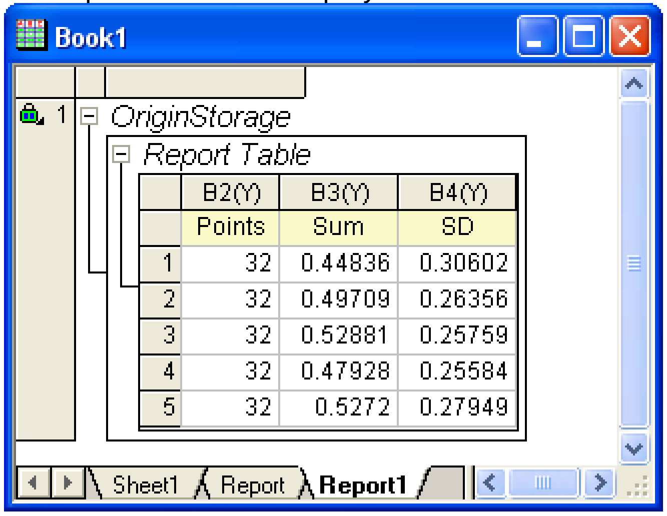
```
// Get data from Data Range object

matrix mData;

if( rng.GetData(mData) <= 0 )

{

        XF_THROW("No data is selected");

}



// Calculate the basic statistics on each column

vector vPoints, vSum, vSD;

for(int index = 0; index < mData.GetNumCols(); index++)

{
        vector vData;

        mData.GetColumn(vData, index);



        int nPoints;

        double dSum, dSD;

        ocmath_basic_summary_stats(vData.GetSize(), vData, 
&nPoints,

                                &dSum, &dSD);



        vPoints.Add(nPoints);

        vSum.Add(dSum);

        vSD.Add(dSD);

}



// Create data table

int nID = 0; // ID should be unique

ReportTable rTable = rt.CreateTable("Report", "Report Table", 
++nID, 0, 1);

rTable.AddColumn(vPoints, "N", ++nID, "Points", 
OKDATAOBJ_DESIGNATION_Y);

rTable.AddColumn(vSum, "Sum", ++nID, "Sum", 
OKDATAOBJ_DESIGNATION_Y);

rTable.AddColumn(vSD, "SD", ++nID, "SD", 
OKDATAOBJ_DESIGNATION_Y);



// Set this attribute as 0 not to specify any format.

// Many bits GETNBRANCH_* defined in oc_const.h to set table 
display format.

rTable.SetAttribute(TREE_Table, GETNBRANCH_TRANSPOSE);
```

4. Create a new worksheet with five columns, then fill all the columns with uniform random data (by highlighting all columns and selecting context menu item **Fill Columns With: Uniform Random Numbers**). Run `OptionStringH -d;` in the Command window to open the dialog. In the dialog, make sure that all the columns are selected for **Input Data** and <*new*> is selected for **Report**. Click the **OK** button and the report sheet will be displayed in the normal format (H:0).



	N(Y)	Sum(Y)	SD(Y)
Long Name			
1	32	0.44836	0.30602
2	32	0.49709	0.26356
3	32	0.52881	0.25759
4	32	0.47928	0.25584
5	32	0.5272	0.27949
6			
7			
8			
9			

5. Open the X-Function in the X-Function Builder again and change *H:0* to *H:1*. Then save it. Run `OptionStringH -d;` in the Command window again. This time the report sheet will display in hierarchical table format (H:1).



	B2(Y)	B3(Y)	B4(Y)
	Points	Sum	SD
1	32	0.44836	0.30602
2	32	0.49709	0.26356
3	32	0.52881	0.25759
4	32	0.47928	0.25584
5	32	0.5272	0.27949

18.3.10 I - Restrict the Behavior of Input Range Control

Restricts the input data range and modifies the behavior of the interactive controls. This Option String only works for the following input data types: Range, XYRange, XYRangeComplex, XYZRange, vector, vector<string>, and vector<complex>.

I:0x0001

Allows multiple data selection in the 1st subrange.

I:0x0002

Allows multiple data selection in the 2nd subrange.

I:0x0004

Allows multiple data selection in the 3rd subrange.

I:0x0008

Allows multiple data selection in all subranges.

I:0x0010

(Obsolete) Restricts to one data set. This is default behavior, no need to set.

I:0x0020

Supports Y error, XYRange only.

I:0x0040

Supports label area in Range. By using this Option String, the output range will not clear the data before execution, which is similar to **I:0x00040000**.

I:0x0080

The <*input*> option will disappear from the pop-up menu of the output variable.

I:0x0100

The <*new*> option will disappear from the pop-up menu of the output variable.

I:0x0200

The *(<input>,<new>)* options will disappear from the pop-up menu of the output variable, XYRange only.

I:0x0400

Gets rid of the button for the pop-up menu.

I:0x0800

Gets rid of the interactive button.

I:0x1000

Valid for input variables with vector and Column type only. When the string for data selection is obtained, Columns are identified by indices, but not the short names of the columns.

I:0x2000

This Option String will make the vector variable only use the Y column while initializing.

I:0x4000

Shows the **Column Browser** button.

I:0x8000

Sets variable as read-only, both **I:0x0400** and **I:0x0800** together will also work.

I:0x00010000

Gets rid of row selection while initializing, Column variable only.

<u>I:0x00020000</u>

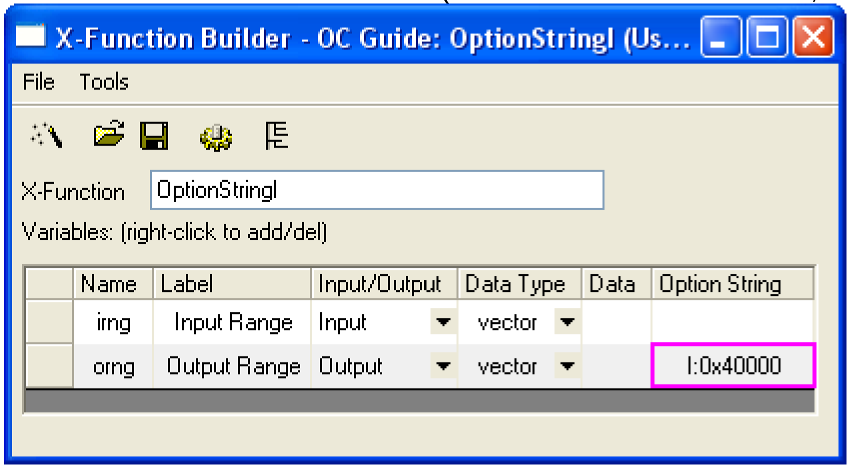
Replaces the range string of the whole sheet with column notation. For example, if a worksheet in *Book1* named *Sheet1* contains two columns, then the range string *[Book1]Sheet1* will be replaced with *[Book1]Sheet1!1:2*.

<u>I:0x00040000</u>

This Option String makes the output range not clear the data before finishing the execution. Without this Option String, if there is no overlap in the input data ranges, columns in the output ranges will be cleared before using new data.

<u>Example</u>

1. Create a new X-Function with the name and variables shown in the following image. Then click the **Save** button to save the X-Function in <*User Files Folder*>*\X-Functions\OC Guide* (if this folder does not exist, create it).



X-Function: OptionStringI

Variables: (right-click to add/del)

Name	Label	Input/Output	Data Type	Data	Option String
irng	Input Range	Input	vector		
orng	Output Range	Output	vector		I:0x40000

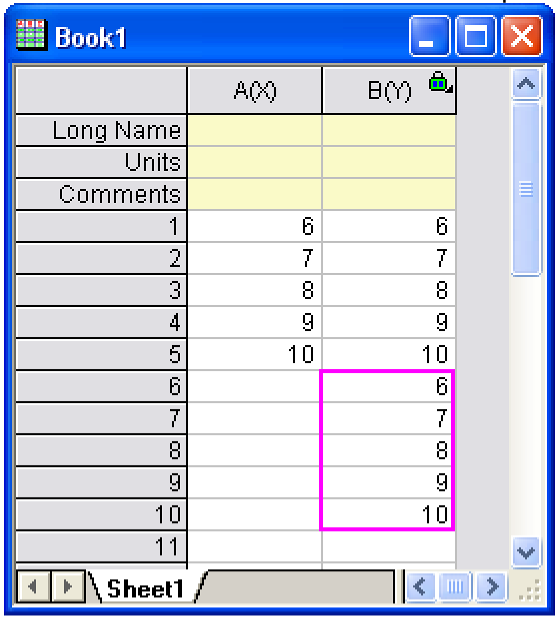
2. Open this X-Function in **Code Builder**, then copy the following code, which will copy data from the input column to the output column, into the body of the main function, **OptionStringI**, and then click the **Compile** button.

```
orng = irng;
```

3. Create a new worksheet with two columns: column A and column B, then select the first 10 rows in column A and choose **Fill Range with: Row Numbers** from the right-click menu to fill these 10 rows with row numbers. Run `OptionStringI -d;` in the Command Window, choose column A as input and column B as output, then click the **OK** button. The result is that column B is now also filled with row numbers.

4. Select the first 5 rows from column A, right-click them to open the context menu, and choose the **Delete** item to delete these 5 rows. The color of the lock on the header of column B changes from green to yellow. Click the lock and choose **Recalculate** from the pop-up menu. The result is that the original data

in the last 5 rows of column B are kept there, as shown in the following image.



Book1

	A(X)	B(Y)
Long Name		
Units		
Comments		
1	6	6
2	7	7
3	8	8
4	9	9
5	10	10
6		6
7		7
8		8
9		9
10		10
11		

I:0x00080000

Makes all columns show in the interactive pop-up menu.

I:0x00100000

If there are no selections in the active graph layer and the range variable is set by using <*active*>, the range variable will set to all plots in the active layer.

I:0x00200000

Uses the sampling interval as output X.

I:0x00400000
I:0x00800000

Makes the existing columns or plots show in the interactive pop-up menu.

I:0x02000000

If the output XYRange is set by using <*auto*>, this option creates a new X column.

I:0x04000000

The row range will be ignored, and only the single block range is valid. If there is no selection or the selection only includes one single cell, the selection will be considered to be the whole worksheet.

I:0x08000000

Creates the range string by using the first and last X values instead of the indices. The syntax is **[BookName]SheetName!ColName[xFirstValue:LastValue]**. It is very useful for monotonous data.

I:0x10000000

The Range variable will have no factor and no weight.

18.3.11 M - Support Multi-line String

M:m-n
Makes a string variable support multi-line text. If the string variable is hidden, *m* lines of text are supported, if expanded, *n* lines are supported.

18.3.12 N - Setting the Default Name of Output Objects

Specifies the default name of an output object.

- **ReportTree** and **ReportData**

N: "Name" Book:="Book Name" Sheet:="Sheet Name"

- **MatrixObject**

N: "Name" Book:="Book Name" Sheet:="Sheet Name" X:="Object Name"

- **MatrixLayer**, **Image**, **vector**, **Column**

N: "Name"

- **XYRange**

N: X:="X Name" Y:="Y Name"

- **XYZRange**

N: X:="X Name" Y:="Y Name" Z:="Z Name"

- **Worksheet**

N: Book:="Book Name" Sheet:="Sheet Name"

18.3.13 O - Setting Output Object Action after Running X-Function

Specifies the action of the output variable after running successfully.

O:A

Activates the corresponding Origin object, no matter if the object is hidden or in a different folder.

O:C

Activates the corresponding Origin object only when the workbook is the active window.

O:N

Does not activate the corresponding Origin object.

18.3.14 P - Setting Control Editable on GUI

Specifies whether a variable is editable in the X-Function dialog.

P:0 or P:1

P:1 is the default Option String. Using this Option String, the output variable, which is of an Origin object type, such as column, worksheet, etc., will be non-editable when bringing up the X-Function dialog again via **Change Parameters**. The variables with other types need to specify the Option String as **P:0** to do so.

P:2

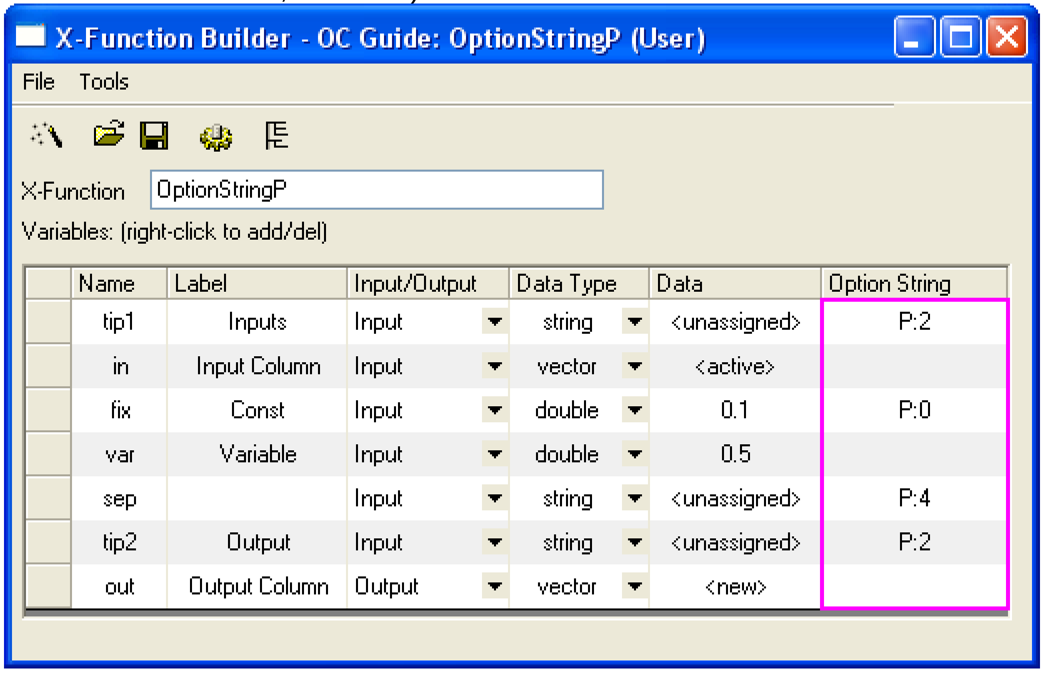
Only shows the label, and the editable field is invisible.

P:4

Shows the variable as a separator in the X-Function dialog; valid for string type only.

Example

1. Press F10 to open X-Function Builder, and create a new X-Function with the name and variables shown in the following image. Then click the **Save** button to save the X-Function in *<User Files Folder>\X-Functions\OC Guide* (if this folder does not exist, create it).



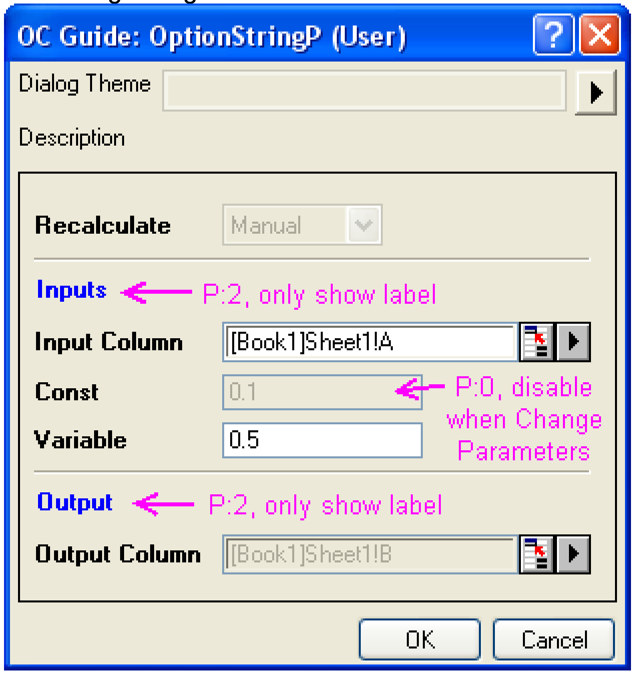
X-Function: OptionStringP

Variables: (right-click to add/del)

Name	Label	Input/Output	Data Type	Data	Option String
tip1	Inputs	Input	string	<unassigned>	P:2
in	Input Column	Input	vector	<active>	
fix	Const	Input	double	0.1	P:0
var	Variable	Input	double	0.5	
sep		Input	string	<unassigned>	P:4
tip2	Output	Input	string	<unassigned>	P:2
out	Output Column	Output	vector	<new>	

2. Click the **Edit X-Function in Code Builder** button to open this X-Function in **Code Builder**. Type the code `out = in + var;` into the body of the main function, **OptionStringP**, and then click the Compile button Compile.
3. Create a new worksheet and fill some data into column A. Run `OptionStringP -d;` in the Command window. In the dialog that opens, select column A as **Input Column** and column B as **Output Column**. Click the **OK** button, and the result is generated and stored in column B.
4. Click the lock icon on the header of column B and choose **Change Parameters** from the context menu. The difference between the dialog opened by **Change Parameters** and the one opened by `OptionStringP -d;` is shown in the

following image.



18.3.15 R - Restrict the Behavior of Dialog Combo Box

Used to control the value of the combo box in the X-Function when calling it via LabTalk script. It will not affect the dialog.

R:0

No restriction. Besides the values defined in the list in the combo box, other values can also be assigned to the combo box variable by using LabTalk script.

R:1

Default Option String. Only the values defined in the list in the combo box can be assigned to the combo box variable by using LabTalk script.

Example

1. Create a new X-Function (named *OptionStringR*) with a variable (named *x1*) of int type. In the **Control** column, set *AA*|*BB*|*CC* for this variable.
2. Run `OptionStringR x1:=5;` in the Command window, and you will get an error: **#Command Error!**.
3. Open this X-Function in X-Function Builder again, set **R:0** for the int variable in the **Option String** column and save this X-Function.
4. Run `OptionStringR x1:=5;` in the Command window again, and you will get no error.

18.3.16 S - Defining Default Data Selection for Input Variable

Defines the default data object for input variable. This Option String is only for variables of Range type.

S:0

All columns in the active worksheet are selected as the data range.

S:0x01

The first matrix object of the active matrix sheet, or the first column of the active worksheet, is selected as the data range.

S:0x10

Makes it so the <*active*> option gets nothing but the matrix object. For the output variables, the <*active*> option will be replaced with <*new*>, which will create a matrix page.

18.3.17 SV - Support String Item in X-Function Dialog Combo Box

Replaces the item values in a combo box with special values. For example, if a variable of int type is set as a combo box via the *Begin|Mid|End* string in the **Control** column, then the default item values for the combo box items are 0, 1, and 2, respectively. If the Option String *SV:1|5|-1* is used for this variable, the return values of the items in the combo box are 1, 5, and -1, respectively.

18.3.18 T - Skipping the Variable from Theme

Controls whether a variable is remembered in the X-Function dialog theme file. This Option String is only valid during theme selection in Tree View, which is the default setting.

T:0

If the specified theme is selected, the theme value will be used as the variable value.

T:1

Does not use the theme value as the variable value, even if a theme is selected.

T:4

If the specified theme is selected, both the theme value and the attribute will be used by the variable. This option is mainly used on Origin internal objects, such as XYRange, Image, etc.

18.3.19 U - Specifying Output as Optional

Specifies whether the output variable is needed. This option string is only available for X-Functions with multiple output variables.

U:1

Adds a selected check box for the output variable. This is the default option.

U:0

Adds an unselected check box for the output variable.

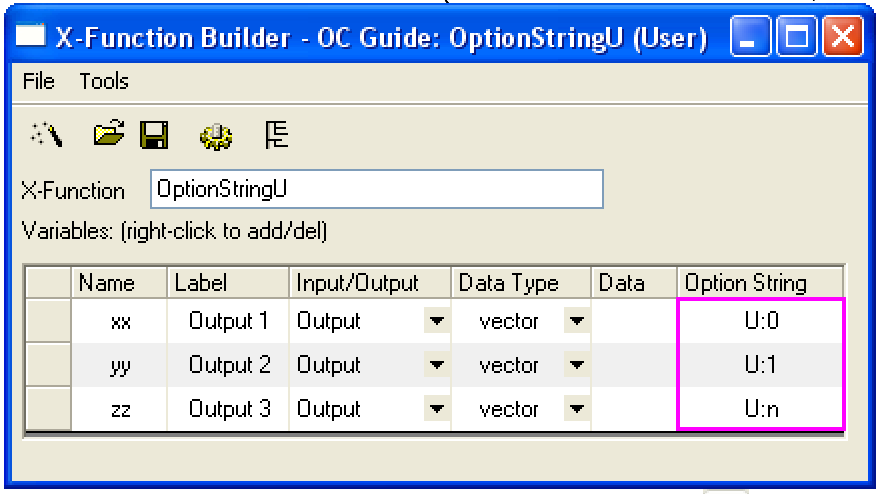
U:n

Adds no check box for the output variable.

Example

1. Create a new X-Function with the name and variables shown in the following image. Then click the **Save** button 💾 to save the X-Function in <*User Files*

Folder>\X-Functions\OC Guide (if this folder does not exist, create it).



X-Function: OptionStringU

Variables: (right-click to add/del)

	Name	Label	Input/Output	Data Type	Data	Option String
	xx	Output 1	Output	vector		U:0
	yy	Output 2	Output	vector		U:1
	zz	Output 3	Output	vector		U:n

2. Click the **Edit X-Function in Code Builder** button to open this X-Function in **Code Builder**. Copy the following code into the body of the main function, **OptionStringU**.

```
vector vv;

vv.Data(1,10,1);



if( xx ) // to check if output xx variable

{

        xx = vv;

}



if( yy ) // to check if output yy variable

{

        yy = vv;

}



zz = vv;
```

3. Put the following code into the **OptionStringU_event1** function, which will get the check box status of the output variables. Then click the Compile button Compile.

```
foreach(TreeNode subnode in trGetN.Children)

{

        string strVarName = subnode.tagName;

        int nUStatus, nOutput;


        // to check if is output variable

        if((subnode.GetAttribute(STR_XF_VAR_IO_ATTRIB, nOutput)

                && IO_OUTPUT == nOutput))

        {

                // check Output check box status


        if(subnode.GetAttribute(STR_ATTRIB_DYNACONTROL_USE_CHECK,
nUStatus))

                {

                        switch(nUStatus)

                        {

                        case 1:

                                printf("%s variable has Output
checked check box.\n"

                , strVarName);

                                break;

                        case 0:

                                printf("%s variable has Output
unchecked check box.\n"
```

```
                , strVarName);
                                break;
                        }
                }
                else
                {
                        printf("%s variable NOT Output check 
box.\n", strVarName);
                }
        }
}
```

4. Run `OptionStringU -d;` in the Command window, keep the default settings of the dialog, and click the **OK** button. This will create a new worksheet with two columns, and the check box status of each output variable will be printed in the Command window.

18.3.20 V - Specifying Control to be Invisible

Specifies whether a variable is visible or not in the X-Function dialog.

V:0

The variable will be invisible in the dialog.

V:1

The variable will be visible in the dialog.

V:2

The variable will be invisible when using Labtalk script.

18.3.21 Z - Adding Check Box to Set Editable or Disable for Variable Control

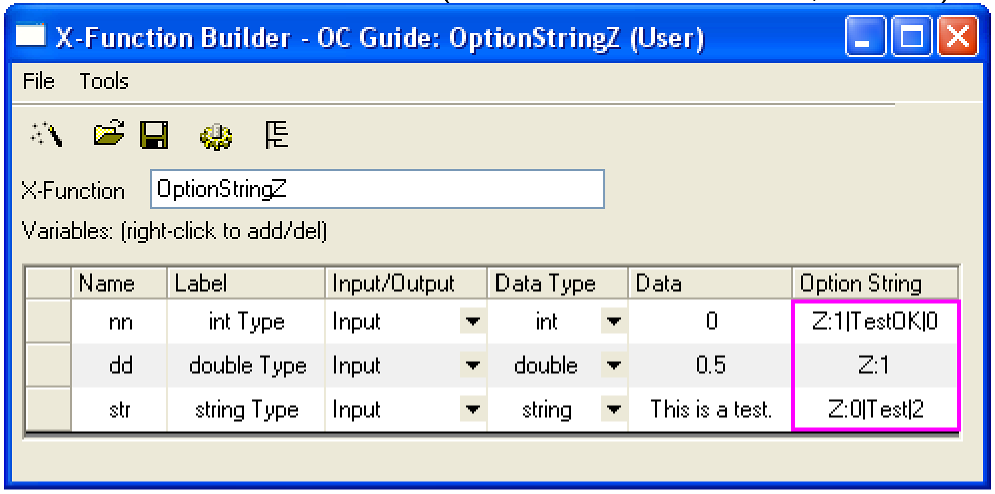
Adds a check box beside an input variable to specify whether it is editable or not. The syntax is **Z:*State*|*Label*|*Behavior***.

- ***State***: *0* or *1*, to specify the status of the check box.
- ***Label***: label for the check box. If both ***Label*** and ***Behavior*** are unspecified, the label is **Auto** by default.
  - ***Behavior***:
    0, the variable is always editable, no matter whether the check box is checked or not.
    1, if checked, the input variable is disabled.

2, if unchecked, the input variable is disabled.

Example

1. Create a new X-Function with the name and variables shown in the following image. Then click the **Save** button to save the X-Function in *<User Files Folder>\X-Functions\OC Guide* (if this folder does not exist, create it).



X-Function: OptionStringZ

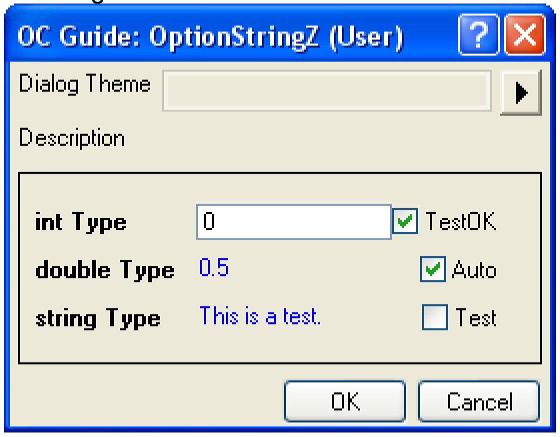
Variables: (right-click to add/del)

Name	Label	Input/Output	Data Type	Data	Option String
nn	int Type	Input	int	0	Z:1|TestOK|0
dd	double Type	Input	double	0.5	Z:1
str	string Type	Input	string	This is a test.	Z:0|Test|2

2. Click the **Edit X-Function in Code Builder** button to open this X-Function in **Code Builder**. Copy the following code into the **OptionStringZ_event1** function, which will get the check box status of the **int Type** variable. Then click the Compile button Compile.

```
TreeNode trInt = trGetN.nn;

int nAutoType = octree_get_auto_support(&trInt);

switch(nAutoType)

{

case -1:

        out_str("Auto check box of int Type control is
unchecked");

        break;

case 0:

        out_str("No auto check box for int Type control");

        break;
case 1:
        out_str("Auto check box of int Type control is checked");
        break;
}
```

3. Run `OptionString -d;` in the Command window, and a dialog like the one in the image below will open. Change the status of the **TestOK** check box, and a message about the status of this check box will output in the Command window.



int Type 0 ✓ TestOK
double Type 0.5 ✓ Auto
string Type This is a test. ☐ Test

Index