Origin C Programming Guide

Copyright © 2026 by OriginLab Corporation

All rights reserved. No part of the contents of this book may be reproduced or transmitted in any form or by any means without the written permission of OriginLab Corporation.

OriginLab, Origin, and LabTalk are either registered trademarks or trademarks of OriginLab Corporation. Other product and company names mentioned herein may be the trademarks of their respective owners.

OriginLab Corporation

One Roundhouse Plaza

Northampton, MA 01060

USA

(413) 586-2013

(800) 969-7720

Fax (413) 585-0126

Table of Contents

1 Introduction to Origin C 1

2 Language Fundamentals 5

3 Predefined Classes 31

4 Creating and Using Origin C Code 43

5 Matrix Books Matrix Sheets and Matrix Objects 67

6 Workbooks Worksheets and Worksheet Columns 91

7 Graphs 141

8 Working with Data 187

9 Projects 201

10 Importing 217

11 Exporting 231

12 Analysis and Applications 237

13 Output Objects 289

14 Accessing Database 293

15 Accessing LabTalk 299

16 Accessing X-Function 305

17 Calling Python Functions from Origin C 309

18 User Interface 313

19 X-Functions 387

20 Accessing External Resources 475

21 Reference 505

Index 513

1 Introduction to Origin C

Origin provides two programming languages: Origin C and LabTalk.

This guide covers the Origin C Programming language. It also shows you how to create X-Functions and control Dialog Builder dialogs. X-Functions provide a framework within Origin to create tools. Dialog Builder allows you to create and control custom dialogs such as floating tools, dialog boxes and wizards.

This guide should be used in conjunction with the Language Reference help files accessible from the Origin Help menu.

The most up-to-date source of documentation including detailed examples can be found at our wiki site: wiki.OriginLab.com.

1.1 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.

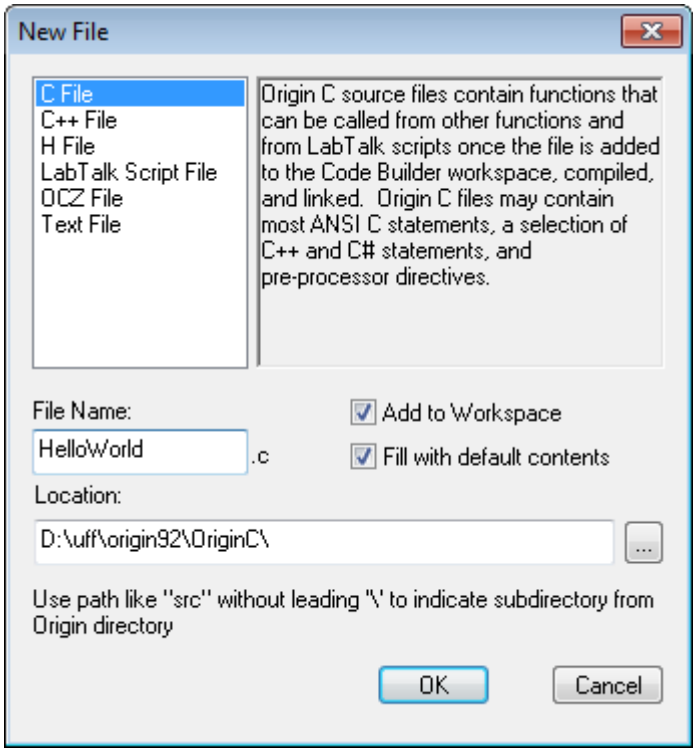
1.2 Hello World Tutorial

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.

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

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

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



Click **OK** and the new file will be opened in Code Builder's Multiple Document Interface (MDI).

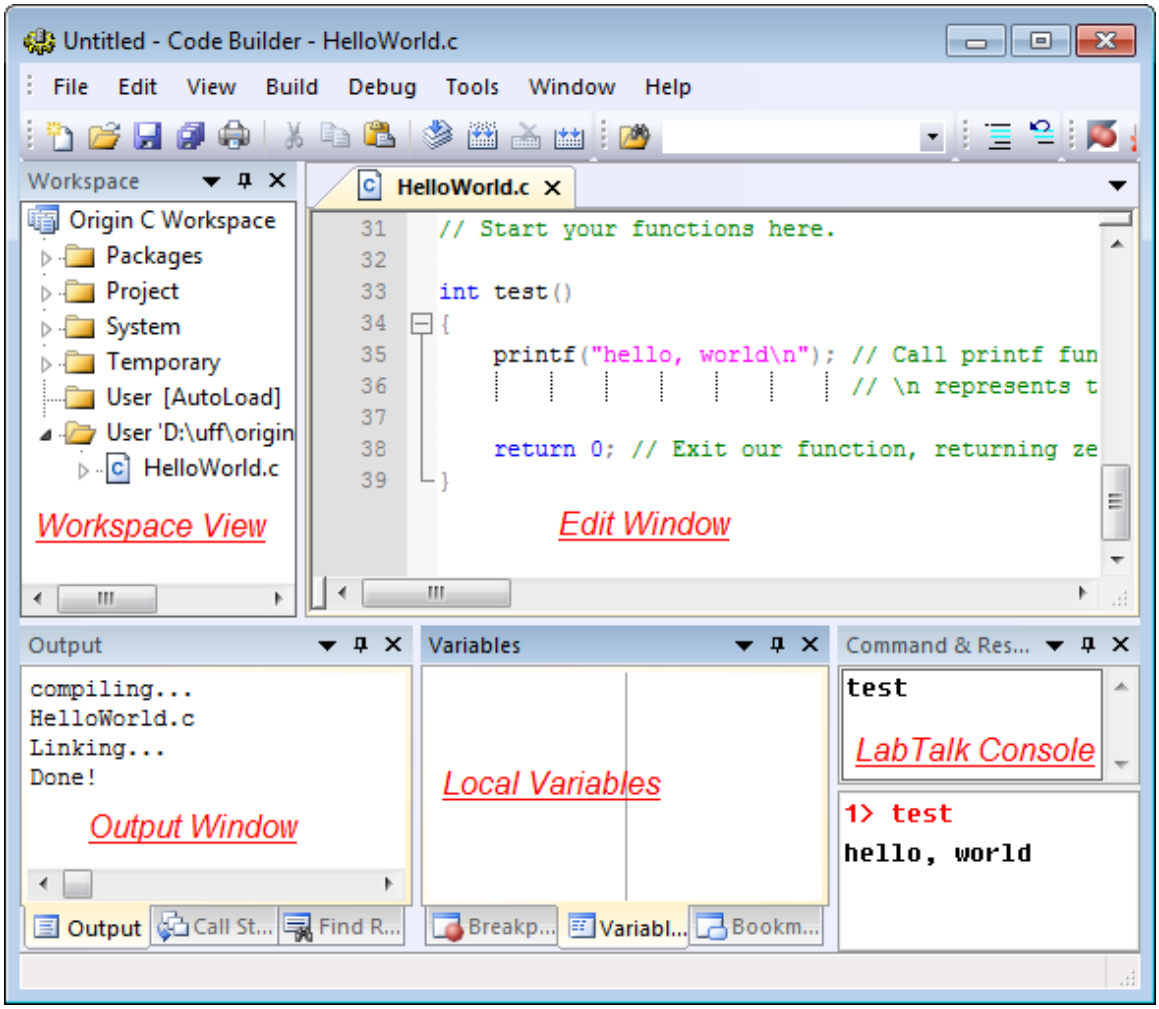
Copy or type the following Origin C code beneath the line that reads *// Start your functions here*.

```
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
}
```

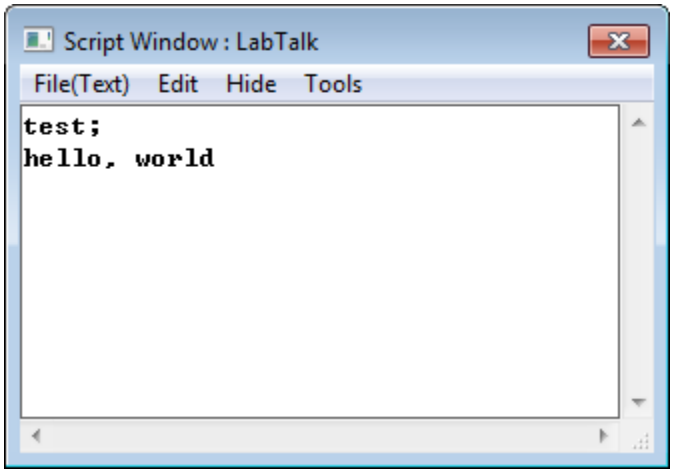
Click the **Build** button  on Code Builder's Standard toolbar to compile and link the *HelloWorld.C* source file.

The **Output** window of **Code Builder** should display as



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.

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.



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

2.1 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.

This section covers the following topics:

Data Types and Variables

Operators

Statement Flow Control

Functions

Classes

Error and Exception Handling

2.2 Data Types and Variables

2.2.1 ANSI C Data Types

Origin C supports all the ANSI C : char , short , int , float , double , and void . In addition, you can declare arrays of, and pointers to, each of these 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
```

2.2.2 Origin C Composite Data Types

Although C-style array syntax is supported, Origin C provides the **string** , **vector** , and **matrix** classes to simplify working with one- and two-dimensional data. Supported element types include char, byte, short, word, int, uint, and complex. A **vector** can be of type string (string array), but a **matrix** cannot; matrix is numeric 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};     // 'double' implied

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

matrix<int> mA1;                       // Matrix of integers
matrix mA2;                            // Matrix of doubles (implied)

// NOTE: If no element type is specified, vector/matrix default to double.
```

Another useful class is **complex**. It stores a real and an imaginary component.

```
complex cc(4.5, 7.8);          // real = 4.5, imaginary = 7.8
out_complex("value = ", cc); // Output the complex value
```

2.2.3 Color

Colors in Origin C are represented by a **DWORD** value. This can be either an index into Origin's internal color palette or an RGB color composed of red, green, and blue components.

2.2.3.1 Palette Index

Origin's internal contains 24 colors. A palette index is zero-based (0–23). Origin C provides named constants, each beginning with **SYSCOLOR_**.

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;
```

2.2.3.2 Auto Color

A special palette index, **Auto**, colors an element using the same color as its parent (when supported by that element's UI).

Use the **INDEX_COLOR_AUTOMATIC** macro to specify Auto:

```
DWORD dwColor = INDEX_COLOR_AUTOMATIC;
```

2.2.3.3 RGB

An Origin color can also represent an RGB value (8-bit red, green, and blue components). Use the **RGB** macro to create an RGB value, then convert it to an Origin color with **RGB2OCOLOR**.

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

To test whether a color is RGB vs. palette index, use **OCOLOR_IS_RGB**:

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

To extract component values from an RGB-based Origin color, first get the RGB with **GET_CRF_FROM_RGBOCOLOR**, then query components:

```
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));
}
```

2.3 Operators

Operators in Origin C support the same arithmetic, logical, comparison, and bitwise operators as ANSI C. The sections below list the four operator types and show usage.

2.3.1 Arithmetic Operators

Operator	Definition	Example

*	multiplication	`10 * 3 -> 30`
/	division	`6 / 2 -> 3`
%	modulus (remainder)	`11 % 2 -> 1`
+	addition	`5 + 4 -> 9`
-	subtraction	`9 - 4 -> 5`
^	exponentiation (power) See note below.	`10 ^ 3 -> 1000` (Origin C default)

Note. By default, Origin C treats the caret (^) as the —the exponentiation (power) operator—to match LabTalk. In standard ANSI C, ^ is the bitwise XOR operator. To use ^ as XOR in Origin C, add `#pragma xor` before your code (e.g., `#pragma xor(push, FALSE)` … `#pragma xor(pop)`).

```
out_int("10 raised to the 3rd is ", 10^3); // default: ^ is exponent in Origin C

#pragma xor(push, FALSE)                   // interpret ^ as XOR
out_int("10 XOR 3 is ", 10^3);
#pragma xor(pop)                           // restore default

// Portable exponentiation that never needs pragma:
out_double("pow(10,3) is ", pow(10,3));
```

Dividing an integer by another integer yields an integer result by default. Use the pragma below so the compiler treats numeric literals as double.

```
int a = 3, b = 2;
```

```
out_double("a/b           -> ", a/b);           // 1 (integer division)
out_double("3/2           -> ", 3/2);           // 1 (integer division)
out_double("3/2.0         -> ", 3/2.0);         // 1.5 (double literal)
out_double("(double)3/2   -> ", (double)3/2);   // 1.5 (explicit cast)

// Treat numeric literals as double so 3/2 becomes 1.5:
#pragma numlittype(push, TRUE)
out_double("3/2 with numlittype -> ", 3/2);     // 1.5

// Variables still follow normal promotions; these now become double
out_double("3/b with numlittype  -> ", 3/b);    // 1.5
out_double("a/2 with numlittype  -> ", a/2);    // 1.5
#pragma numlittype(pop) // restore default
```

The 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);
```

A common pattern is to test odd/even:

```
int n = 27;
out_str(n % 2 ? "27 is odd" : "27 is even");
```

2.3.2 Comparison Operators

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

Operator	Definition	Example

>	greater than	`5 > 4 -> 1`
>=	greater than or equal to	`5 >= 5 -> 1`
<	less than	`3 < 2 -> 0`
<=	less than or equal to	`2 <= 2 -> 1`
==	equal to	`7 == 7 -> 1`
!=	not equal to	`7 != 8 -> 1`

```
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");
```

2.3.3 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 (short-circuit).

Operator	Definition	Example
!	NOT	`!0 -> 1`

&&	AND	`1 && 0 -> 0`
||	OR	`0 || 1 -> 1`

Consider the following two examples:

```
expr1A && expr2

expr1B || expr2
```

`expr2` will not be evaluated if `expr1A` evaluates to false or `expr1B` evaluates to true. Ordering can prevent errors and improve efficiency. For example, guard against divide-by-zero:

```
int denom = 0, num = 10;

// Safe: right side never executed because denom == 0 makes whole && false

if ( denom != 0 && (num / denom) > 2 )

    out_str("greater than 2");

// Reversing the order could evaluate num/denom first and fail.
```

Another classic short-circuit guard uses a NULL check:

```
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 `upperLimit` because reading the member from a NULL pointer can cause an application to end abruptly.

2.3.4 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	Definition	Example
~	bitwise complement (invert bits)	`~5 -> 250` (00000101 → 11111010, 8-bit)
<<	shift left	`1 << 3 -> 8` (00000001 → 00001000)
>>	shift right	`8 >> 2 -> 2` (00001000 → 00000010)
&	bitwise AND	`6 & 3 -> 2` (00000110 & 00000011 = 00000010)
^	bitwise exclusive OR (XOR) See note below.	`5 ^ 3 -> 6` (00000101 ^ 00000011 = 00000110; requires `#pragma xor`)
|	bitwise inclusive OR	`5 | 2 -> 7` (00000101 | 00000010 = 00000111)

Note. By default, Origin C treats ^ as exponentiation (power). Use `#pragma xor` to interpret ^ as bitwise XOR instead:

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

Common bit-mask patterns:

```
// Bit flags
const int FLAG_READ  = 0x01;  // 0001
const int FLAG_WRITE = 0x02;  // 0010
const int FLAG_EXEC  = 0x04;  // 0100
```

```
int perms = 0;
perms |= FLAG_READ;                 // set
perms |= FLAG_WRITE;                // set
out_int("perms after set    = ", perms); // 0x03

perms &= ~FLAG_WRITE;               // clear
out_int("perms after clear  = ", perms); // 0x01

perms ^= FLAG_EXEC;                 // toggle
out_int("perms after toggle = ", perms); // 0x05

out_str( (perms & FLAG_READ) ? "READ on" : "READ off" ); // test
```

2.4 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** statement for looping through a collection of objects.

2.4.1 The if Statement

The **if** statement tests a condition and executes a block when the test is true. The **if-else** statement executes an alternative block when the test is false.

Examples using different input types:

```
bool bb = true;    // Boolean
if( bb )
{
    out_str("bb is true");
```

```
}


int nn = 5;

if( nn )            // Integer: 0 = false, non-zero = true

{

    out_str("nn not 0");

}


double* pData = NULL;

if( NULL == pData ) // Pointer check

{

    out_str("Pointer pData is NULL");

}
```

An **if-else** block:

```

bool bRet = true;

if( bRet )

{

    out_str("Valid input");      // when bRet is true

}

else

{

    out_str("INVALID input");    // when bRet is false

}
```

Braces are optional if the block has only one statement:

```

```

```
if( bRet )
    out_str("Valid input");
else
    out_str("INVALID input");
```

An **else if** ladder (common pattern):

```
int score = 86;
if( score >= 90 )
    out_str("Grade: A");
else if( score >= 80 )
    out_str("Grade: B");
else if( score >= 70 )
    out_str("Grade: C");
else
    out_str("Grade: D/F");
```

2.4.2 The switch Statement

Use **switch** to execute different blocks for mutually exclusive choices. The `break` statement exits the switch block (a `break` after `default` is optional but harmless).

```
// Using an enum makes intent clearer
enum DataType { TYPE_INT = 1, TYPE_FLOAT = 2, TYPE_STRING = 3, TYPE_OTHER = 4 };

int nType = TYPE_FLOAT;
switch( nType ) // integer value as condition
{
case TYPE_INT:
case TYPE_FLOAT:
```

```
    out_str("Numeric type");
    break;

case TYPE_STRING:
    out_str("String type");
    // no break here, fall through to TYPE_OTHER

case TYPE_OTHER:
    out_str("Other type");
    break;

default:
    out_str("Unrecognized type");
    break; // optional
}
```

2.4.3 The for Statement

The **for** statement runs a fixed number of times or steps through an array by index.

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

Iterating over a vector by index and by element:

```
```

```
vector<double> v = {1.2, 3.4, 5.6};


// Index-based
for( int i = 0; i < v.GetSize(); i++ )
    printf("v[%d] = %g\n", i, v[i]);


// Element-based using foreach over the collection
foreach(double val in v)
    printf("val = %g\n", val);
```

2.4.4 The while Statement

while and **do-while** execute a block until a condition is met. **while** tests at the beginning; **do-while** tests at the end.

```
int count = 0;
while( count < 10 )
{
    out_int("count = ", count);
    count++;
}
```

A simple menu loop with **do-while** (executes at least once):

```
int choice;
do
{
    out_str("1) Import  2) Analyze  3) Exit");
    choice = 3; // substitute a real input mechanism if needed
```

```
    if( 1 == choice )
        out_str("Importing...");
    else if( 2 == choice )
        out_str("Analyzing...");
}
while( choice != 3 ); // exit on 3
```

2.4.5 Jump Statements

Jump statements unconditionally transfer control within a function: **break**, **continue**, and **goto**.

2.4.5.1 break

The **break** statement exits the nearest enclosing loop or switch immediately.

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

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

2.4.5.2 continue

The **continue** statement skips the rest of the current loop iteration and proceeds with the next iteration.

```
printf("The odd numbers from 1 to 10 are:\n");
for( int index = 1; index <= 10; index++ )
{
    if( (index % 2) == 0 )       // use % (modulus) to test even/odd
        continue;                // skip even numbers
```

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

Another **continue** use: skip blanks when counting letters.

```
const char* s = "A B  C";
int letters = 0;
for( int i = 0; s[i]; i++ )
{
    if( ' ' == s[i] )
        continue;
    letters++;
}
out_int("letters = ", letters); // 3
```

2.4.5.3 goto

The **goto** statement jumps to a labeled statement within the same function. Use sparingly; structured control flow is usually clearer.

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

out_str("Skipped statement");

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

2.4.6 The foreach Statement

Use **foreach** to loop through a collection. The example below iterates all pages in the project and prints 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 collection-based classes in Origin C.

2.5 Functions

2.5.1 Global Functions

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

Basic IO

Character and String Manipulation

COM

Communications

Curve

Data Conversion

Data Range

Date Time

File IO

File Management

Fitting

Image Processing

Import Export

Internal Origin Objects

LabTalk Interface

Math Functions

Mathematics

Matrix Conversion and Gridding

Memory Management

NAG

Signal Processing

Spectroscopy

Statistics

System

Tree

User Interface

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

2.5.2 User-Defined Functions

Origin C supports user-defined functions. A user-defined function lets you create functions that accept your choice of arguments and return type, and operate on those arguments to achieve a specific purpose.

2.5.2.1 Minimal example (pass by value)

Define the function and call it in one block:

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

// Usage
double d = 3.3;
d = my_function(d);
```

```
out_double("d == ", d); // 13.3
```

2.5.2.2 Modify a value (pass by reference)

Use a reference parameter to modify the caller's variable without returning a value.

```
void add_offset(double& x, double offset)
{
    x += offset; // modifies 'x' in the caller
}

// Usage
double v = 5;
add_offset(v, 2.5);
out_double("v == ", v); // 7.5
```

2.5.2.3 Overloading and default arguments

Origin C supports function overloading and default parameter values.

```
// Two overloads for scale(): vector<double> and vector<int>
void scale(vector<double>& v, double factor = 1.0)
{
    int n = v.GetSize();
    for (int i = 0; i < n; i++)
        v[i] *= factor;
}

void scale(vector<int>& v, double factor = 1.0)
{
    int n = v.GetSize();
```

```
    for (int i = 0; i < n; i++)
        v[i] = (int)(v[i] * factor);
}

// Usage
vector<double> a = {1.0, 2.0, 3.0};
scale(a, 2.0);              // -> {2, 4, 6}

vector<int> b = {1, 2, 3};
scale(b);                   // default factor 1.0 (no change)
```

2.5.2.3.1 Copy-return variant (read-only input)

Keep the input read-only and return a scaled copy.

```
vector<double> scale(const vector<double>& v, double factor)
{
    vector<double> out = v;       // copy input
    int n = out.GetSize();
    for (int i = 0; i < n; i++)
        out[i] *= factor;
    return out;
}

// Usage
vector<double> c = {2.5, 4.0};
vector<double> d2 = scale(c, 0.5); // c unchanged, d2 = {1.25, 2.0}
```

2.5.2.4 Calling from LabTalk

You can call eligible Origin C functions from LabTalk (parameters/return types must meet certain criteria).

```
// Example function callable from LabTalk:
double add_then_scale(double x, double y, double factor)
{
    return (x + y) * factor;
}

// In the Script Window or LabTalk Console:
double r;
r = add_then_scale(2.5, 3.5, 10); // calls the Origin C function
r=; // outputs 60
```



For details on which signatures are callable from script and how to keep functions available, see

the LabTalk help on calling Origin C functions.

2.6 Classes

Origin C supports many built-in classes, and also allows you to create your own.

2.6.1 Origin Defined Classes

Origin C includes predefined classes for working with Origin data types and user-interface objects. These classes help you quickly write Origin C code to accomplish common tasks. This section introduces the base classes to give you an overview of their capabilities. See the next chapter, *Predefined Classes*, or the Origin C Wiki for details and examples of Origin-defined classes.

2.6.2 User Defined Classes

Origin C supports user-defined classes. A user-defined class lets you create objects of your own type with methods (member functions) and data members.

The example below defines a **Book** class with:

a constructor to set the name on creation,

a **const** getter (safe to call on const objects),

and two setter overloads (from C-string or from *string*).

```
class Book
{
public:
    // Constructors
    Book() : m_strName("") {}                    // default
    Book(LPCSTR name) : m_strName(name) {}       // set name on creation

    // Getter (const so it can be called on const Book)
    string GetName() const
    {
        return m_strName;
    }

    // Setters (overloaded for convenience)
    void SetName(LPCSTR lpcszName)
    {
        m_strName = lpcszName;
    }
    void SetName(const string& name)
    {
        m_strName = name;
    }

private:
```

```
    string m_strName;

};
```

And here is a simple usage example that declares a **Book** object, sets its name, and outputs it:

```
void test_class()

{

    Book one;                    // default constructor

    one.SetName("ABC");

    out_str(one.GetName());



    const Book two("Origin C");  // construct with name; const object

    out_str(two.GetName());      // OK (GetName is const)

    // two.SetName("new");       // NOT allowed: 'two' is const

}
```

For more class features—such as constructors/destructors in larger examples or virtual methods—download this zip file, then browse to *\Origin C Examples\Programming Guide\Extending Origin C* to see *EasyLR.c*, *EasyLR.h*, and *EasyFit.h*.

2.7 Error and Exception Handling

2.7.1 Exception Handling

Origin C supports C++-style exception handling using the **try**, **catch**, and **throw** statements .

A **try** block is the keyword `try` followed by statements in braces. Immediately after the try block is a single **catch** handler.

Note:** Origin C supports only **one** catch handler and it must accept an *int* error code.

```
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);
}
```

During execution, statements in the try block run first. If an error is thrown, control jumps to the catch block; otherwise the catch block is skipped.

You can use **throw** to signal an error explicitly and jump to the catch block:

```
void TryCatchThrowEx()
{
    try
    {
        DoSomeWork(4);   // valid -> prints result
        DoSomeWork(-1);  // invalid -> throws
    }
    catch (int iErr)
    {
        printf("Error code = %d\n", iErr);
    }
}

void DoSomeWork(double num)
{
    if ( num < 0 )
```

```
        throw 100; // force error: negative input

    if ( 0 == num )

        throw 101; // force error: zero not allowed



    double result = sqrt(num) / log(num);

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

}
```



Because only `catch(int)` is supported, choose and document distinct integer codes for different error conditions.

3 Predefined Classes

3.1 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.

This section covers the following topics:

Analysis Class

Application Communication Class

Composite Data Types Class

Internal Origin Objects Class

System Class

User Interface Controls Class

Utility Class

3.2 Analysis Class

3.2.1 Analysis Class

The following classes are used to perform data analysis. For details, see the **Origin C Reference: Classes – Analysis** chapter in the Origin C help.

Class	Brief Description
NLFitContext	Provides access to information about the fitting function and to the current evaluation state generated when implementing a fitting function in Origin C.
NLFitSession	Higher-level wrapper around the NLFit class that offers a friendlier interface for implementing the fitting evaluation procedure. It underlies the NLFit dialog and is recommended for Origin C coding because it handles much of the interfacing

	complexity.

3.3 Application Communication Class

3.3.1 Application Communication Class

The following classes enable communication between Origin and other applications. For details, see the **Origin C Reference: Classes – Application Communication** chapter in the Origin C help.

Class	Brief Description
Matlab	Enables communication between Origin and MATLAB.

3.4 Composite Data Types Class

3.4.1 Composite Data Types Class

The following classes are composite data-type classes. For details, see the **Origin C Reference: Classes – Composite Data Types** chapter in the Origin C help.

Class	Brief Description
CategoricalData	An array of integers tied to an internal Origin **Text** data set. Values map text to category indices (1-based). The mapped texts live in the associated **CategoricalMap**.
CategoricalMap	An array of unique text values (allocated dynamically), not tied to an internal data set. Values are stored alphanumerically and referenced by **CategoricalData** indices.
complex	Handles complex numbers; stores both real and imaginary parts and provides complex arithmetic/utilities.

Curve	Derived from **curvebase** and **vectorbase**. Represents a Y data set with an optional associated X. Can be plotted via **GraphLayer** methods.
curvebase	Abstract base of curve-like classes (used polymorphically). Not constructible; use a derived class such as **Curve**.
Dataset	Dynamic array derived from **vector**/**vectorbase**. May be tied or not tied to an internal Origin data set. Default element type is **double**; supports basic types (char, byte, short, word, int, uint, complex; not string). Use `Dataset<type>`.
Matrix	Two-dimensional array derived from **matrix**/**matrixbase**, tied to an Origin matrix window. Default type is **double**; supports basic types (char, byte, short, word, int, uint, complex; not string). Use `Matrix<type>`. Accesses matrix data; **MatrixObject** controls display/style. Worksheet "Z" values map linearly to X (columns) and Y (rows).
matrix	Two-dimensional array (lowercase *m*) derived from **matrixbase**, not tied to a matrix window (more flexible). Default type is **double**; supports the same basic types (not string). Use `matrix<type>`.
matrixbase	Abstract base for **matrix** and **Matrix** (polymorphic use). Not constructible; use derived classes.
PropertyNode	Container for typed properties (bool, int, float, double, string, vector, matrix, picture, etc.).
string	Null-terminated character array (similar to MFC *CString*) with rich string APIs. Works with `vector<string>` for string arrays.
Tree	Save/load Origin C trees as XML.
TreeNode	Build multi-level trees; traverse nodes; access attributes.

TreeNodeCollection	Retrieve collections of child nodes with a shared name prefix.
vectorbase	Abstract base for **vector** and **Dataset** (polymorphic use). Not constructible; use derived classes.
vector	Dynamic array derived from **vectorbase**, not tied to an internal data set (more flexible). Default type is **double**; supports basic types including char, byte, short, word, int, uint, complex, and string. Use `vector<type>`.

3.5 Internal Origin Objects Class

3.5.1 Internal Origin Objects Class

The following classes are used to handle Origin objects. For details, see the **Origin C Reference: Classes – Internal Origin Objects** chapter in the Origin C help.

Class	Brief Description
Axis	Access axes contained in layers on an Origin page.
AxisObject	Access axis objects (ticks, grids, labels) contained by axes on a graph page.
Collection	Generic container template for internal Origin objects (e.g., `Collection<PageBase> Pages`). Supports attach/detach and idiomatic looping (e.g., `foreach`).
CollectionEmbeddedPages	Access pages embedded in a worksheet.
Column	Wrapper for worksheet columns; each holds a **Dataset**. Primarily controls display/style of the associated data.

DataObject	Base class for worksheet columns and matrix objects; data objects live in layers on a page.
DataObjectBase	Abstract base for **DataObject** and **DataPlot** hierarchies (polymorphic use).
DataPlot	Wrapper for graph data plots; stores plot characteristics and resides in a graph layer.
DataRange	Build/query ranges from Worksheet/Matrix/Graph. Holds page, sheet/layer, and index info; can aggregate multiple sub-ranges.
DataRangeEx	Extension of **DataRange**.
DatasetObject	Access non-numeric datasets (typically members of **Column**).
Datasheet	Handle worksheet and matrix layers.
Folder	Project Explorer folder wrapper; provides access to pages and folder hierarchy.
fpoint3d	3D point with *double* (x, y, z).
fpoint	2D point with *double* (x, y).
GetGraphPoints	Get screen/data (x, y) positions from a graph window.
GraphLayer	Wrapper for graph layers; contains a collection of **DataPlot** objects; owned by a **GraphPage**.
GraphObject	Handle graph objects (text/graphic annotations, style holders, ROI objects). Contained by a **GraphLayer**.

GraphPage	Wrapper for graph pages (windows); provides access to layers and their objects.
GraphPageBase	Base class for **GraphPage** and **LayoutPage**.
Grid	Format data-sheet windows; selection, labels, cell color, merge, etc.
GroupPlot	Handle grouped plots contained in a layer.
Layer	Wrapper for page layers; many objects on a page are contained in layers.
LayoutPage	Wrapper for layout pages (windows); exposes page methods and layer access.
Layout	Wrapper for layout layers within a layout page.
MatrixLayer	Handle matrix layers; contains a collection of matrix objects in the layer.
MatrixObject	Wrapper for matrix objects; controls style/display. Data access is via **Matrix**; analogous to **Column** (style) vs **Dataset** (data).
MatrixPage	Wrapper for matrix pages (windows); provides access to matrix layers and their objects.
Note	Wrapper for Note pages (windows).
OriginObject	Base class for all Origin objects.
Page	Wrapper for general pages; contains one or more layers (except Note).
PageBase	Abstract base for page types; useful for generic functions or attaching to

	the active page.
point	2D integer point (x, y).
Project	Access most objects in the project; exposes collections (pages, datasets) and active objects (ActiveCurve/Layer/Folder).
ROIObject	Handle region-of-interest objects, typically on matrices.
Scale	Handle axis scales (each graph layer has X and Y scales).
storage	Save binary (**TreeNode**) and INI (**INIFile**) data into Origin objects.
StyleHolder	Wrapper for data-plot style holders (plot type info).
UndoBlock	Safe project edits via `UndoBlockBegin()`/`UndoBlockEnd()`.
WorksheetPage	Wrapper for worksheet pages (windows).
Worksheet	Handle worksheet layers; contains a collection of **Column** objects.
XYRange	Build/get XY ranges (one X, one Y). Holds page/sheet/layer and index info; can contain multiple sub-ranges.
XYRangeComplex	XY ranges for complex data (matrix/worksheet); multiple sub-ranges supported.
XYZRange	XYZ ranges (matrix/worksheet); multiple sub-ranges supported.



Many of these classes are **wrappers** around internal Origin objects (they reference an internal object; multiple wrappers can refer to the same internal object). Where applicable, data access vs. style/display responsibilities are split (e.g., **Matrix** for data, **MatrixObject** for style).

3.6 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.

3.7 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.
*DhtmlControl	//add description here
*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.

3.8 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 Creating and Using Origin C Code

This section covers the following topics:

Create and Edit an Origin C File

Compiling, Linking and Loading

Debugging

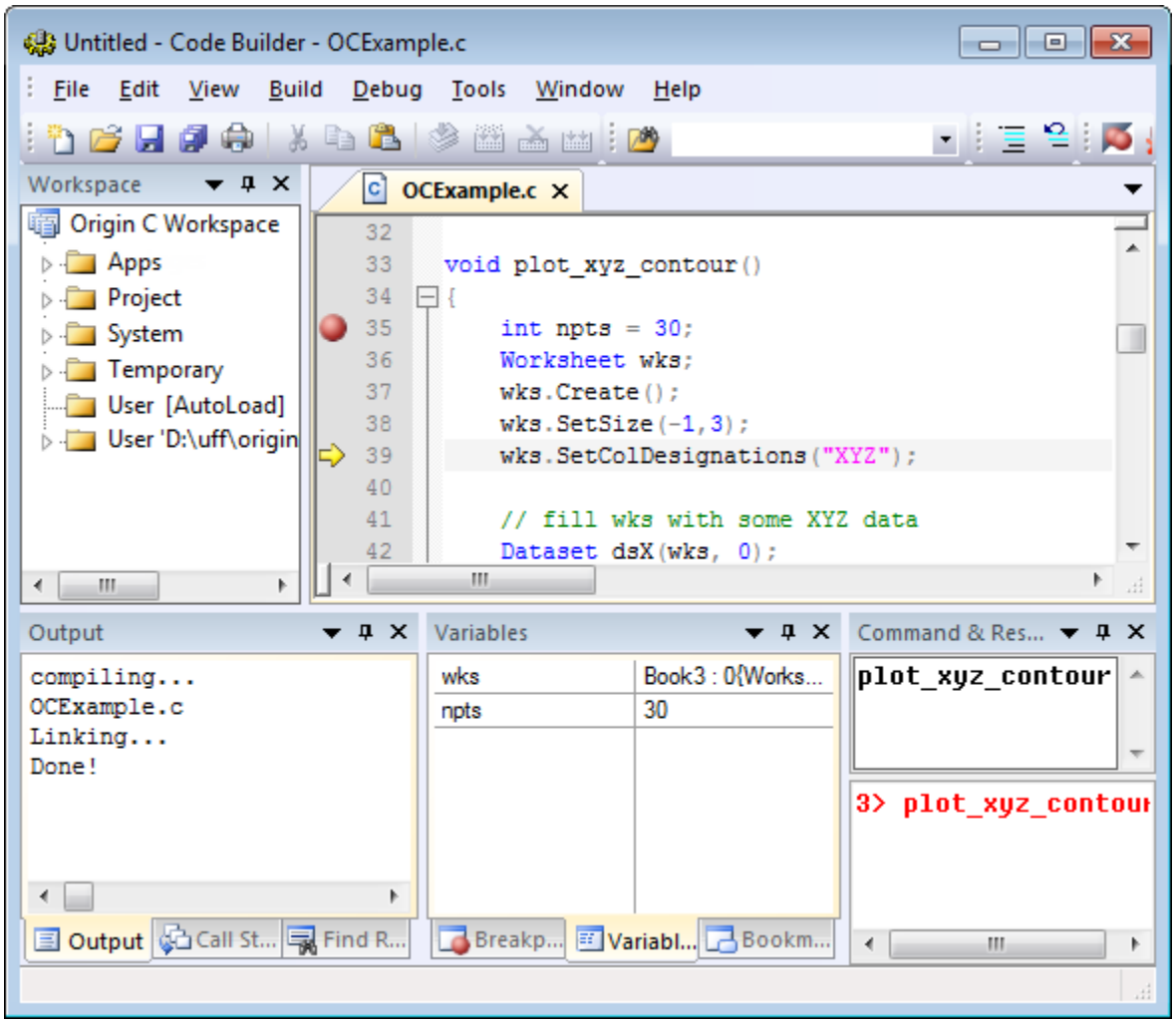
Using Compiled Functions

Distributing Origin C Code

4.2 Create and Edit an Origin C File

4.2.1 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.



The Code Builder window

4.2.2 File Types

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

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

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. Code Builder also allows you to create an encrypted Origin C source file (*.ocz) so it can be safely shared with others.

Until source files have been compiled, linked and loaded, the functions they contain cannot be used in Origin.

4.2.2.2 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.

4.2.2.3 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 and @OCSB. 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
```

4.2.2.3.1 @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.

4.2.2.3.2 @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.



Notes:

1. The generated OP and OCB have 32 bit and 64 bit versions. For example, the op file generated from abc.c file on a 32 bit version will be named as abc_32.OP.

2. Since Origin 9.0, the generated 32 bit or 64 bit version file works only in its

corresponding version (32 bit or 64 bit) of Origin.

4.2.2.4 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.

4.2.3 The Workspace View

The Code Builder Workspace view contains six folders:

Apps

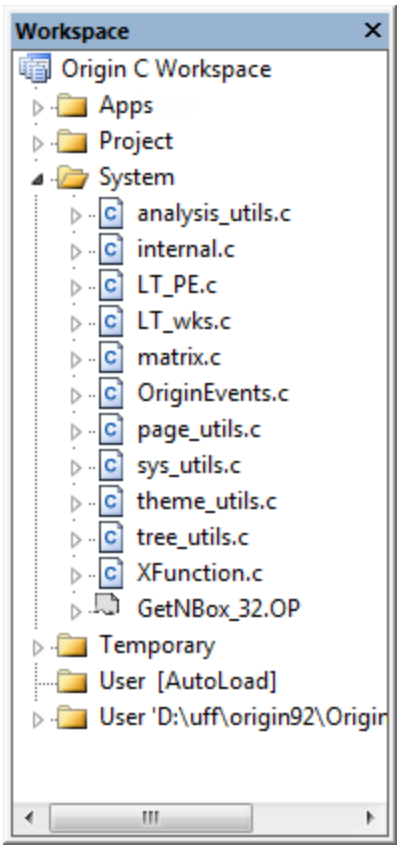
Project

System

Temporary

User [AutoLoad]

User



The Workspace View

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

4.2.3.1 Apps

This folder is used to manage packages. This folder contains only folders, and each folder represents a disk folder in Apps Folder.

Context menu of "Apps" folder

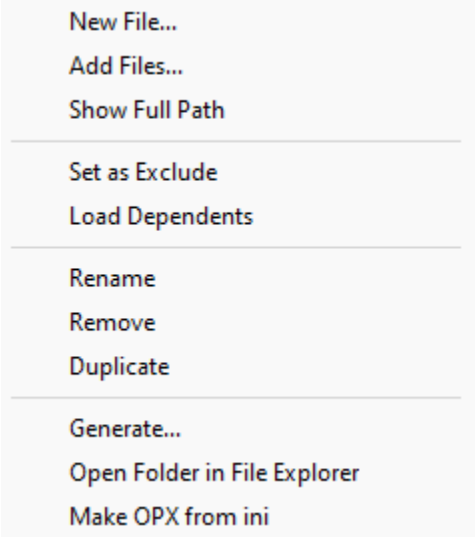
When you right click on the Apps folder, there is a context menu with two items:

Add Existing Folder...

New

The first is for choosing a folder that already exists in the Apps Folder, and the second is for creating a new folder named **Untitled** which also creates a new disk folder named **Untitled** in Apps Folder. Repeating **New** will create enumerated **Untitled** folders.

Context menus of each package folder.



New File...

This is used to create file in the folder.

Add Files

This is used to add files to the folder. Each package folder represent a disk folder in Apps Folder. If selected files are not already in the Apps Folder\packageFolder\, they will be copied to Apps Folder\packageFolder\ folder.

Show Full Path

Show or hide the full path of the files.

Set as Exclude

Prevents all files in an App folder from being compiled and linked when the Build button is clicked in Code Builder.

Load Dependents

Load and compile all dependent Origin C files into the Temporary folder.

Rename

Rename the folder.

Remove

Remove the folder from Code Builder. This will not actually delete the folder from the file system.

Duplicate

Duplicate the folder.

Generate...

Launch the Package Manager and add the files from the package folder. If an OPX in the same file system location as the last time it was saved exists, that OPX will be loaded and all files removed before adding all the files from the package folder.

Open Folder in File Explorer

Open package folder in file explorer.

Make OPX from ini

Generate the OPX file by "package.ini" if exists. See [1]

4.2.3.2 Project

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.

4.2.3.3 System

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.

4.2.3.4 Temporary

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.

4.2.3.5 User [AutoLoad]

This folder is similar with the **User Folder** described below, except that the files in this folder will be compiled and linked automatically when Origin is started, and then the functions defined in the files under this folder are available, and no need to compile and link manually.

4.2.3.6 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.



Notes:

The contents of the **Apps** and **User [AutoLoad]** folders persist across all Origin sessions, while the contents of the **Project** folder are unique to each Project file (OPJ).

4.2.4 Code Builder Quick Start

Get started using Code Builder in just a few steps:

Open Code Builder by pressing **Alt+4** on the keyboard or by clicking the Code Builder toolbar button .

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.

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");
}
```

You need to compile and link the code before calling it. You can compile and link it by pressing **Shift+F8** or by clicking the Build toolbar button .

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.

Click in the top part of the Command & Results window. Since no argument is needed for the function, type either **HelloWorld** or **HelloWorld()** and press Enter. In the bottom part of the Command & Results window you should see a repeat of 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.

4.3 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.

4.3.1 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.

4.3.1.1 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.

4.3.1.2 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.

4.3.1.3 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.

4.3.1.3.1 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.

4.3.2 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:

4.3.2.1 Add files to the Project Folder

When you add files to Code Builder's **Project folder**, they will be built automatically each time the associated Origin project is opened.

You can add files to the **Project** 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.

4.3.2.2 Add files to the User [AutoLoad] Folder

Formerly, you could right-click on the **System** folder in Code Builder and **Add Files**. Since Origin 2015, you are prevented from adding user files to this folder. When you want your files to be compiled and linked automatically on Origin startup, you should add these files to the **User [Autoload]** folder.

You can add files to the **User [AutoLoad]** folder using the following methods:

Right-click on the User [AutoLoad] folder (in Code Builder) and choose **Add files**.

Drag a file from another Workspace folder and drop it on the User [AutoLoad] folder.

Using the **run.LoadOC()** object.

Note: When you add files to the **User [AutoLoad]** folder, a section called **[OriginCAuto]** is added to the Origin.ini file in your User Files Folder. To simultaneously remove files from **User [AutoLoad]** and remove the added **[OriginCAuto]** from your Origin.ini, run this LabTalk command:

```
run -cra // does not affect System folder files
```

4.3.2.3 Build Workspace on Origin Startup

The **Build on Startup** option will build the most recently opened Code Builder workspace upon 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** folder by enabling the **Build on Startup** option.

Run 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.

4.3.2.4 Build Individual Source File on Origin Startup

The following steps show how to modify the Origin.ini to load and compile Origin C source files on startup.

Make sure Origin is not running. Open the Origin.ini file in your User Files Folder(type "%Y="<Enter> in the Script or Command window to locate your User File Folder).

In the **[Config]** section, uncomment (remove the leading ";") in the line under the [Config] section, so that it is as below:

```
Ogs1 = OEvents
; Ogs2 = OEvents
; Origin can trigger multiple system events
; Uncomment this line and implement event handlers in OEvents.ogs
```

Save and close this file.

Open OEvents.ogs under the Origin installation folder. Find the **[AfterCompileSystem]** section and add the following line as a new line

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

Save OEvents.ogs to your User Files Folder (same location as the Origin.ini file that you edited) and close your editor.

Restart Origin and open Code Builder. In the Temporary folder, there are 3 files. AscImpOptions depends on fu_utils.c and Import_utils.c, so the compiler compiles AscImp, along with the two files. For more details please search **run.LoadOC** in your Labtalk documentation.

Alternately, use the ***User [AutoLoad]*** folder in your Workspace. Files added in the folder will be automatically loaded on Origin startup (see above).

4.3.3 Building by Script

Origin C source file must be compiled and linked before calling its functions in LabTalk script. If you don't want to add the source file into Code Builder do compile and linke, use the LabTalk command *Run.LoadOC* to compile and link it. For example:

Choose File->New Workspace... to create a new workspace. The Temporary folder should be empty now.

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.

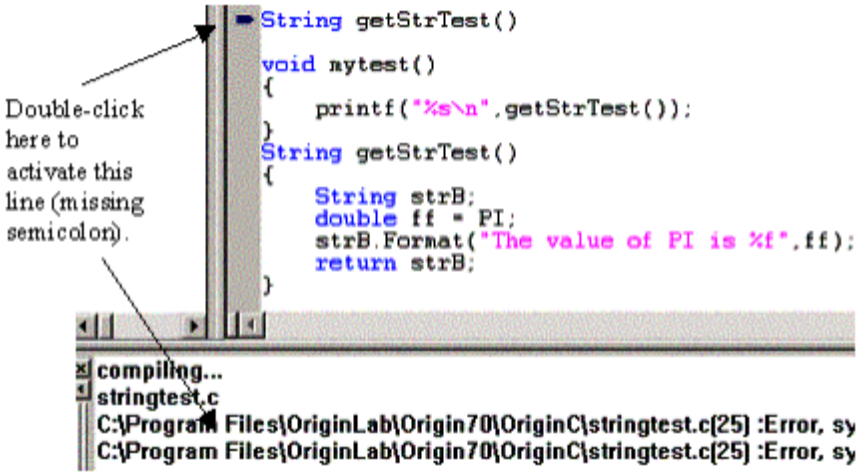
```
if(run.LoadOC(OriginLab\dragNdrop.c, 16) != 0)
{
        type "Failed to load dragNdrop.c!";
        return 0;
}
```

4.3.4 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.



4.4 Debugging

4.4.1 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.



When debugging Origin C code, you can hover on a variable name with your mouse to see a tooltip with the variable's current value.

When a variable has a more complex structure, the tooltip can display a tree-like structure. To disable tree display, add the following to OIDEOptions.ini:

```
[Environment]
bTreeLikeDataTip=0
```

4.4.2 Macros for Debugging

Origin C allows users to define multi-parameter macros which have many uses. Many programmers use output statements while developing code to indicate program flow and display the values of variables at key moments.

4.4.2.1.1 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.");

}
```

4.4.2.1.2 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.

4.5 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.

4.5.1 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.

4.5.1.1 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.

4.5.1.2 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.

4.5.1.3 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.

4.5.1.3.1 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);
```

4.5.1.3.2 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);
```

4.5.1.4 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:

LabTalk Function (vector)

Origin C Function

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.

4.5.2 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

4.6 Distributing Origin C Code

4.6.1 Distributing Source Code

Origin users can share Origin C source code with one another by distributing either the source files themselves (.C, .CPP, .OCZ) 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 the encrypted Origin C source files (.OCZ) or preprocessed files (.OP) for version before Origin 2015 instead of the source files (.C or .CPP).

For encrypted OCZ files, users only need to drag and drop them into Code Builder in Origin to view and edit content. A prompt will show up to ask for the password when you try to open it for the first time but it will be only asked once in same Origin session.

See the File Types in the **Create and Edit an Origin C File** section for more information.



When an encrypted OCZ file is open in Origin session, since Origin 2016 SR0 user can choose to re-save the *.ocz file as not encrypted *.c or *.cpp by selecting menu **File: Save As** and choose a file type in **Save as type** drop-down list.

4.6.2 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.

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();

}
```

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))

        {

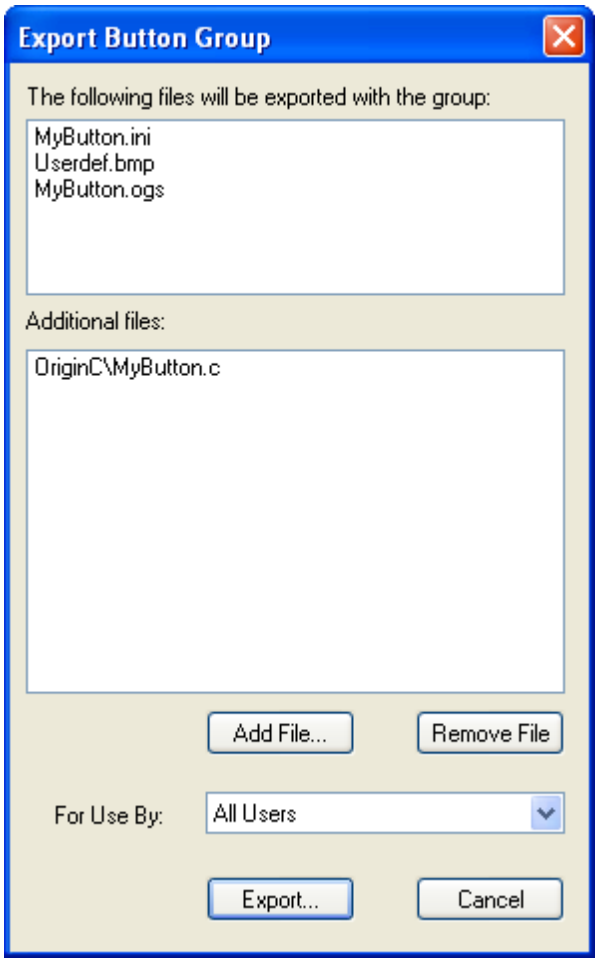
                OnButtonClick;

        }
```

In the Origin menu, choose **View: Toolbars**. In the **Customize Toolbar** dialog, choose the **Button Groups** tab, and click **New** button 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.

In the **Customize Tool** dialog, select *MyButton* item from **Groups** list, and click to choose the  button from **Buttons** panel, then click **Settings** button from **Button** group to open a **Button Settings** dialog. Choose *MyButton.ogs* as the **File Name**, type "Main" in for **Section Name**, then make sure the following check-boxes are unchecked: **Matrix**, **Excel**, **Graph**, **Layout** and **Excel**. Click OK to close the dialog.

Click **Export** to open the **Export Button Group** dialog, then click **Add File** and choose the above *MyButton.c* file.



Click **Export** button, then in the Save As dialog click Save to save the *MyButton.OPX* file to the specified folder.

Choose menu **Tools: Package Manager**, and in the dialog that opens, choose **File: Open** to open the *MyButton.OPX* file. Put the script

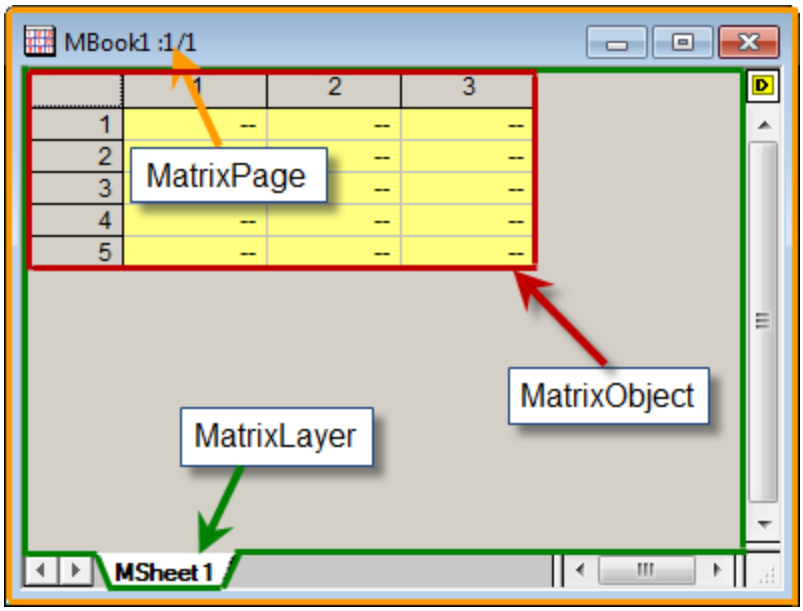
```
Run.LoadOC(%Y\OriginC\MyButton.c);
```

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 Matrix Books Matrix Sheets and Matrix Objects

5.1 Matrix Books Matrix Sheets and Matrix Objects

The Origin C MatrixPage class is for working with Origin matrix books. Each matrix book contains a collection of MatrixLayers and each matrix layer contains a collection of MatrixObjects.



This section covers the following topics:

Base Matrix Book Operation

Matrix Sheets

Matrix Objects

5.2 Base Matrix Book Operation

The Origin C MatrixPage class provides methods and properties common to Origin matrix books. This class is derived from Page class, from which it inherits its methods and properties. And matrixbook has the same data structure level with WorksheetPage in Origin, both are windows. So, they contain lots of similar operations.

5.2.1 Workbook-like Operations

Both matrixbook and workbook are windows, and they share lots of similar operations, and the Basic Workbook Operation chapter can be referred to.

Create New MatrixBook

The same Create method is used.

```
MatrixPage matPg;

matPg.Create("Origin"); // create a matrixbook using the Origin template
```

Open Matrixbook

The difference to open a matrixbook by Open method is that the extension of a matrixbook is *ogm*.

Access Matrixbook

There are multiple ways to access an existing matrixbook and the methods used are the same as workbooks.

The Project class contains a collection of all the matrixbooks in the project. The following example shows how to loop through them.

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

    out_str(matPg.GetName()); // output matrixbook name
```

You can also access a matrixbook 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 matrixbook

    out_str(matPg.GetName()); // output matrixbook name
```

If the matrixbook name is known, this matrixbook can be accessed by passing its name to the class constructor.

```
MatrixPage matPg("MBook1");

if(matPg) // if there is a matrixbook named "MBook1"

    matPg.SetName("MyBook1"); // rename the matrixbook
```

Save Matrixbook

The methods SaveToFile will be used for saving matrixbook as *.ogm file.

```
MatrixPage matPg("MBook1");
```

```
// Save matrixbook as OGM file

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

Show or Hide Matrixbook

This is the same as workbook's show and hide by using the Show property derived from OriginObject class.

Activate Matrixbook

To activate a workbook, the method SetShow can be used by passing parameter of value PAGE_ACTIVATE, which is the same as to activate a workbook.

```
MatrixPage matPg("MBook1");

matPg.SetShow(PAGE_ACTIVATE);  // Activate the matrixbook
```

Delete Matrixbook

The Destroy method can also be used to destroy (delete) a matrixbook.

```
MatrixPage matPg;

matPg = Project.MatrixPages.Item(0); // get first matrixbook in project

if( matPg ) // if there is a matrixbook

    matPg.Destroy(); // delete the matrixbook
```

Clone/Duplicate Matrixbook

The Clone method is also used to clone the matrix page.

```
// Duplicate "MBook1" window with data and style

// Before calling make sure these windows exist

MatrixPage matPage("MBook1");

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

Name and Label Matrixbook

To handle with matrixbook's short name, Long Name and Comments, Origin C provides the same ways as handling workbook's, including the inherited methods SetName, SetLongName, SetComments, and Label property.

5.2.2 Show Image Thumbnails

To show or hide image thumbnails, the method MatrixPage::ShowImageThumbnails is available.

```
MatrixPage mp("tangent");

mp.ShowImageThumbnails(true);  // Pass true to make thumbnail visible
```

5.3 Matrix Sheets

5.3.1 Matrix Sheets

Origin C provides the MatrixLayer class for working with a matrix sheet.

This section covers the following topics:

Basic Matrix Sheet Operation

Matrix Sheet Data Manipulation

5.3.2 Basic Matrix Sheet Operation

Examples in this section are similar to those found in the Basic Worksheet Operation section, because matrix sheet and worksheet are at the same level in the Origin object structure.

5.3.2.1 Add New Matrix Sheet

Add a matrixsheet in a matrixbook using the AddLayer method.

```
// Access the matrixbook named "MBook1"
MatrixPage mp("MBook1");

// Add a new sheet to the matrixbook
int index = mp.AddLayer("New Matrix Sheet");

// Access the new matrixsheet
MatrixLayer mlayerNew = mp.Layers(index);
```

5.3.2.2 Activate a Matrixsheet

To make a matrixsheet in matrixbook to be activated, the function set_active_layer can be used.

```
// Access a matrixsheet by full name
MatrixLayer mLayer("[MBook1]MSheet1");

// Set this matrixsheet to be active
set_active_layer(mLayer);
```

5.3.2.3 Delete Matrixsheet

Use the Destroy method to delete a matrixsheet.

```
MatrixLayer ly = Project.ActiveLayer();
if( ly ) // If the active layer is a matrixsheet
    ly.Destroy(); // Delete the matrixsheet
```

5.3.2.4 Access Matrixsheets in Matrixbook

Similar to accessing worksheets in workbook, matrixsheets in matrixbook can also be accessed by the following ways.

By full layer name.

```
// Full matrixsheet name
string strFullName = "[MBook1]MSheet1!";

// Construct a matrixsheet instance and attach it to the named sheet
MatrixLayer matLy1(strFullName);

// Attach an existing matrixsheet instance to the named sheet
matLy2.Attach(strFullName);
```

A matrixbook constains a collection of matrix layers. Loop through all matrix layers in a specified matrixbook using the **foreach** statement.

```
MatrixPage matPage("MBook1");
foreach(Layer ly in matPage.Layers)
    out_str(ly.GetName());
```

Access a specified matrixsheet by its name or index.

```
// Assume there are at least two matrixsheets on the page MBook1,
// and they are named MSheet1 and MSheet2 separately.
MatrixPage matPage("MBook1");
MatrixLayer lyFirst = matPage.Layers(0); //by index
MatrixLayer lySecond = matPage.Layers("MSheet2"); //by name
```

5.3.2.5 Modify Matrixsheet Properties

5.3.2.5.1 Get and Set Dimensions

In Origin, all matrix objects in matrixsheet share the same dimension (the same number of columns and rows).

To get number of rows and columns in a matrixsheet, you can get the first matrix object of a matrixsheet, and then use the methods (GetNumCols and GetNumRows) in MatrixObject class.

```
// get num rows and cols
MatrixLayer ml = Project.ActiveLayer(); // Get active matrixsheet
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
```

To set dimensions of a matrixsheet, you can use the MatrixLayer::SetSize method.

```
// set num rows and cols
MatrixLayer ml = Project.ActiveLayer(); // Get active matrixsheet
ml.SetSize(-1, 5, 5);  // Set dimensions by 5x5
```

Also, the MatrixObject class has provided the SetSize method for setting dimensions. However, please note, even this method is defined in MatrixObject, what it changes is the matrixsheet's dimension, because all matrix objects in the same matrixsheet have the same dimensions.

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

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

Matrices have numbered columns and rows which are mapped to linearly spaced X and Y values. You can use the SetXY method to set the XY mapping coordinates. Note: this method is available by matrix object, however, the XY mapping is shared by all matrix objects in the same matrixsheet.

```
MatrixLayer ml = Project.ActiveLayer();  // Get active layer
MatrixObject mo = ml.MatrixObjects(0);  // Get the first matrix object
mo.SetXY(-10, 20, -2.3, 12.4);  // Set X from -10 to 20, and Y from -2.3 to
12.4
```

5.3.2.5.2 Get and Set 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 matrixsheet, 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 matrixsheet

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 matrixsheet 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 matrixsheet

// Note, get XY labels from matrixsheet, 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 matrixsheet
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
matrixsheet.

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 matrixsheet

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);
```

5.3.2.5.3 Format Matrixsheet

A matrixsheet can be formatted programmatically using a theme tree.

The example below formats a block of cells in the active matrixsheet 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);
```

5.3.2.5.4 Get and Set 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);
}
```

5.3.3 Matrix Sheet Data Manipulation

5.3.3.1 Conversion Between Matrixsheets and Matrix Objects

In Origin, a matrixsheet can hold multiple matrix objects. Using the matobj_move function, you can split multiple matrix objects into separate matrixsheets, or combine multiple matrixsheets into one (provided all matrices share the same dimensions).

```
// This code snippet is to merge the matrix objects in three sheets to
// a new sheet
MatrixPage mp("MBook1");  // Matrixbook
MatrixLayer ml1 = mp.Layers(1);  // 2nd sheet
MatrixLayer ml2 = mp.Layers(2);  // 3rd sheet
MatrixLayer ml3 = mp.Layers(3);  // 4th sheet

MatrixLayer mlMerge;
mlMerge.Create("Origin");  // Create a new sheet for merging
MatrixObject mo1 = ml1.MatrixObjects(0);  // Matrix object in 2nd sheet
MatrixObject mo2 = ml2.MatrixObjects(0);  // Matrix object in 3rd sheet
MatrixObject mo3 = ml3.MatrixObjects(0);  // Matrix object in 4th sheet
matobj_move(mo1, mlMerge);  // Move the matrix object to the end of the sheet
matobj_move(mo2, mlMerge);
matobj_move(mo3, mlMerge);
```

5.4 Matrix Objects

5.4.1 Matrix Objects

Matrix object, which is MatrixObject class, is the basic unit for storing matrix data, and its container is matrix sheet, that relationship is like column and worksheet. The following pages will show the practical examples on the operation of matrix object.

This chapter covers the following topics:

Basic Matrix Object Operation

Matrix Object Data Manipulation

Converting Matrix to Worksheet

5.4.2 Basic Matrix Object Operation

A matrix sheet can have multiple matrix objects, which share the same dimensions. A matrix object is analogous to a worksheet column and can be added or deleted, etc. The following sections provide some practical examples on the basic operations of matrix object.

5.4.2.1 Add or Insert Matrix Object

It allows to set the number of matrix objects in the matrix sheet by using MatrixLayer::SetSize, so to add matrix objects.

```
// Set 5 matrix objects in the active matrix sheet
MatrixLayer ml = Project.ActiveLayer();
ml.SetSize(5);
```

The method MatrixLayer::Insert will insert a specified number of matrix objects before the current 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
```

5.4.2.2 Activate Matrix Object

To activate a matrix object in the matrix sheet, the MatrixLayer::SetActive is available.

```

```

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

ml.SetActive(2);  // Set 3rd (index is 0-based) matrix object active
```

5.4.2.3 Access Matrix Object

To access a matrix object, you can use the collection of MatrixObjects from MatrixLayer.

```
// 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();

}
```

5.4.2.4 Delete Matrix Object

To delete a specified number of matrix objects from a matrix sheet, you can use the MatrixLayer::Delete method.

```
// 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);
```

5.4.2.5 Switch Between Image Mode and Data Mode

The MatrixLayer::SetViewImage method has provided the option for switching between image mode and data mode of the specified matrix object (by index).

```
// 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);  // FALSE for data view

}
```

5.4.2.6 Get and Set Labels

For each matrix object, you can set Long Name, Comments, and Units. And it actually is to get and set the Z labels, please refer to the Get and Set Z Labels on Base Matrix Sheet Operation chapter.

5.4.2.7 Data Type and Format

5.4.2.7.1 Get and Set Data Type

Matrix object's internal data types include double, real, short, long, char, text, mixed, byte, ushort, ulong, and complex, etc. And Origin C provides the GetInternalDataType and SetInternalDataType methods in MatrixObject class to get and set matrix object internal data type respectively.

```
// 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

}
```

5.4.2.7.2 Get and Set Data Format

The MatrixObject::GetFormat and MatrixObject::SetFormat are provided for getting and setting the data format of a matrix object respectively.

```
// 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 Matrix Object Data Manipulation

5.4.3.1 Set Values 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();
```

5.4.3.2 Copy Matrix Data

The matobj_copy function is used to copy matrix data.

```
MatrixLayer mlSrc = Project.ActiveLayer();  // Get the active matrix sheet

MatrixObject moSrc = mlSrc.MatrixObjects(0);  // Get the 1st matrix object in the
sheet

MatrixLayer mlDst;

mlDst.Create("Origin");  // Create a new matrix sheet

MatrixObject moDst = mlDst.MatrixObjects(0);  // Get the 1st matrix object

bool bRet = matobj_copy(moDst, moSrc);  // Copy the active data to the newly
created matrix
```

5.4.3.3 Math on Matrix Data

To perform mathematical operation on matrix, it always gets the data out of matrix object into a data matrix, and then do the calculation, and put the data back into matrix object. The math includes multiplying matrix by constant, dot multiply, dot divide, dot power, cross, cumulative product, cumulative sum, difference, etc.

The following shows two examples on the matrix operations, one is multiply matrix by constant, and the other is dot multiply.

5.4.3.3.1 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;
```

5.4.3.3.2 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);
```

5.4.3.4 Conversion between Matrix Object and Vector

The methods matrixbase::GetAsVector and matrixbase::SetByVector can be used to convert between matrix object and vector.

```

// To vector
MatrixLayer ml = Project.ActiveLayer();  // Active matrix sheet
MatrixObject mo = ml.MatrixObjects(0);  // The 1st matrix object
matrixbase &mb = mo.GetDataObject();  // Get data from matrix object
vector vb;
mb.GetAsVector(vb);  // Convert the matrix data into vector


// From vector
```

```
MatrixLayer ml1;

ml1.Create("Origin");  // Create a matrix sheet

MatrixObject mo1 = ml1.MatrixObjects(0);  // Get matrix object

matrixbase &mb1 = mo1.GetDataObject();  // Get data object

mb1.SetSize(2, 3);  // Set size 2 rows x 3 columns

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

// Set vector data to matrix object

// First row: 1, 2, 3

// Second row: 4, 5, 6

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

5.4.3.5 Manipulate Matrix Object with Complex Values

Origin C provides a set of methods in matrixbase class for handling complex, including making a complex matrix from two real matrices, getting real and imaginary, getting phase and amplitude, calculating conjugate, etc.

The following code is used to set a matrix object as complex matrix with two real matrices data, and then get its real, imaginary, phase, and amplitude into separate matrix objects, and then use the conjugate to replace the original complex matrix object.

```
void MatrixObject_Complex_EX()

{

        // Original data for real

        matrix mR =

        {

                {2, 2, 2, 0},

                {0, 1, 99, 99}

        };

        // Original data for imaginary

        matrix mI =

        {
```

```
                {3, -3, 0, 3},
                {0, 99, 1, 99}
        };
        matrix<complex> mC;
        // Create a complex data
        int iRet = mC.MakeComplex(mR, mI);
        if(iRet == 0)
        {
                // Create a new matrix sheet for complex data
                MatrixLayer ml;
                ml.Create("Origin");
                MatrixObject mo = ml.MatrixObjects(0);
                ml.SetInternalData(FSI_COMPLEX);
                matrixbase &mb = mo.GetDataObject();
                mb = mC;

                // Get real part
                matrix mReal;
                mb.GetReal(mReal);
                // Get imaginary part
                matrix mImg;
                mb.GetImaginary(mImg);
                // Get phase
                matrix mPha;
                mb.GetPhase(mPha);
                // Get amplitude
                matrix mAmp;
```

```
                mb.GetAmplitude(mAmp);

                // Create new matrix sheet for the results

                MatrixLayer mlRes;

                mlRes.Create("Origin");

                // Set 4 matrix objects, the same size as the matrix

                mlRes.SetSize(4, mb.GetNumRows(), mb.GetNumCols());

                MatrixObject moReal = mlRes.MatrixObjects(0);

                MatrixObject moImg = mlRes.MatrixObjects(1);

                MatrixObject moPha = mlRes.MatrixObjects(2);

                MatrixObject moAmp = mlRes.MatrixObjects(3);

                matrixbase &mbReal = moReal.GetDataObject();

                matrixbase &mbImg = moImg.GetDataObject();

                matrixbase &mbPha = moPha.GetDataObject();

                matrixbase &mbAmp = moAmp.GetDataObject();

                mbReal = mReal;  // Set real part to matrix object

                mbImg = mImg;  // Set imaginary part to matrix object

                mbPha = mPha;  // Set phase to matrix object

                mbAmp = mAmp;  // Set amplitude to matrix object


                // Use the conjugate to replace the original complex matrix

                mb.Conjugate();

        }
}
```

5.4.3.6 Transform Matrix Object Data

Origin C contains a set of methods in matrixbase for the matrix transformation, such as flip a matrix horizontally or vertically, rotate a matrix, shrink a matrix, transpose a matrix, etc.

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

MatrixObject mo = ml.MatrixObjects(0);

matrixbase &mb = mo.GetDataObject();



mb.FlipHorizontal();  // Flip horizontally

mb.FlipVertical();  // Flip vertically

mb.Rotate(90);  // Rotate 90 degrees counter-clockwise, need to be multiple of 90

mb.Shrink(2, 2);  // Shrink by factor of 2 for both row and column

mb.Transpose();  // Transpose
```

5.4.4 Converting Matrix to Worksheet

You may need to re-organize your data by converting from matrix to worksheet, or vice versa, for certain analysis or graphing needs. This page provides information and examples of converting matrix to worksheet, and please refer to Converting Worksheet to Matrix for the "vice versa" case.

5.4.4.1 Matrix to Worksheet

To convert a matrix object data to worksheet, you can firstly get the data in matrix object out to a data matrix, and then use the CopyTo method defined in class.

Here is the example on how to convert the whole matrix object directly into worksheet.

```


// Convert the active matrix object's data into a newly created worksheet directly,

// without tranposing, and with setting the column type the same as matrix

MatrixLayer ml = Project.ActiveLayer();  // Active matrixsheet

MatrixObject mo = ml.MatrixObjects(0);  // Get the first matrix object

matrixbase &mb = mo.GetDataObject();  // Get the data from matrix object

Worksheet wks;

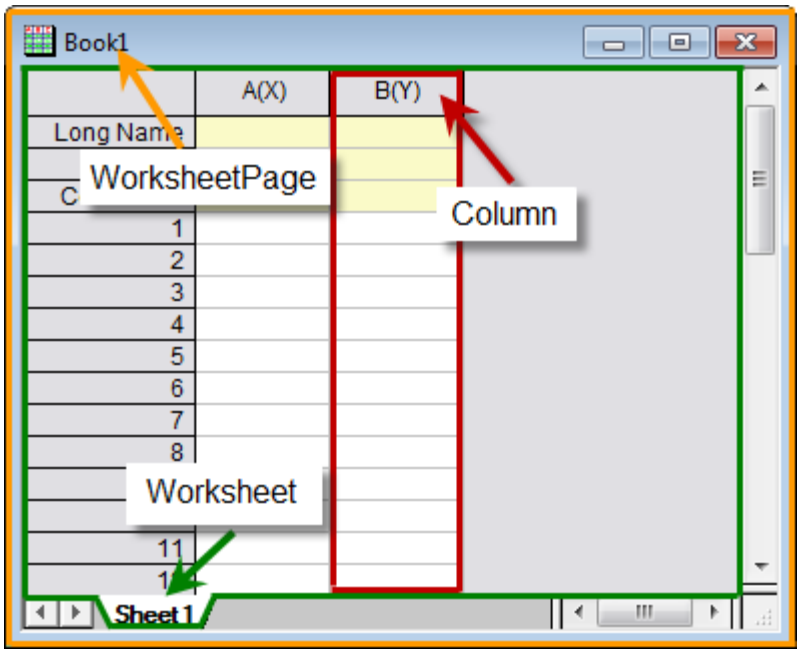
wks.Create("Origin");  // Create a new worksheet

mb.CopyTo(wks, 0, 0, -1, -1, 0, 0, FALSE, TRUE);  // Convert the data to worksheet
```

6 Workbooks Worksheets and Worksheet Columns

6.1 Workbooks Worksheets and Worksheet Columns

The Origin C WorksheetPage class is for working with Origin workbooks. Each workbook contains a collection of Worksheets and each worksheet contains a collection of Columns.



This section covers the following topics:

Workbooks

Worksheet Columns

Worksheets

6.2 Workbooks

6.2.1 Workbooks

The Origin C WorksheetPage class provides methods and properties common to Origin workbooks. This class is derived from Page class, from which it inherits its methods and properties.

This chapter covers the following topics:

Basic Workbook Operation

Workbook Manipulation

6.2.2 Basic Workbook Operation

6.2.2.1 Create New Workbook

The Create method is used for creating new workbooks.

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

6.2.2.2 Open Workbook

If the workbook with data is saved (as extension of ogw), it can be opened by the Open method.

```
Worksheet wks;  // The Open method belongs to Worksheet
string strOGW = "D:\\Book1.ogw";  // Path of the workbook
wks.Open(strOGW);  // Open the workbook
```

6.2.2.3 Access Workbook

There are multiple ways to access an existing workbook. The Project class contains a collection of all the workbooks 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
```

You can also access a workbook by passing its index to the Item method of the Collection class.

```
WorksheetPage wksPg;
```

```
wksPg = Project.WorksheetPages.Item(2);

if(wksPg) // if there is a 3rd workbook

    out_str(wksPg.GetName()); // output workbook name
```

If the workbook name is known, this workbook can be accessed 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
```

6.2.2.4 Save Workbook

Origin allows you to save a workbook with data to a file (*.ogw), or as a template without data (*.otw), and for the workbook with analysis, it is able to be saved as an analysis template (*.ogw). And methods SaveToFile and SaveTemplate are used for saving workbook as *.ogw and *.otw files respectively.

```
WorksheetPage wksPg("Book1");

// Save workbook as OGW file

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

// Save workbook as OTW template

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

// get template name and output

out_str( page_get_template_name(pg) )
```

6.2.2.5 Show or Hide Workbook

The WorksheetPage class inherits the Show property from OriginObject class to show or hide itself.

```
WorksheetPage wksPg("Book1");

wksPg.Show = false;  // Hide the workbook. If true, show the workbook
```

6.2.2.6 Activate Workbook

To activate a workbook, the method SetShow can be used by passing parameter of value PAGE_ACTIVATE.

```
WorksheetPage wksPg("Book1");
wksPg.SetShow(PAGE_ACTIVATE);  // Activate the workbook
// More operations can be done by passing different values, such as
// wksPg.SetShow(PAGE_HIDDEN);  // Hide the workbook
// wksPg.SetShow(PAGE_MINIMIZED);  // Minimize the workbook
// wksPg.SetShow(PAGE_MAXIMIZED);  // Maximize the workbook
```

6.2.2.7 Delete Workbook

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 will destroy it, together with all the sheets in the workbook, and all the columns 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
```

6.2.2.8 Clone/Duplicate Workbook

The WorksheetPage class (for a Workbook) is derived from the Page class. This class has a Clone method that is used to clone the source page.

```
// Duplicate "Book1" window with data and style
// Before calling make sure these windows exist
WorksheetPage wksPage("Book1");
WorksheetPage wksPage1 = wksPage.Clone();
```

6.2.2.9 Name and Label Workbook

For a workbook, there will be short name, Long Name, and Comments. The inherited methods, SetName, SetLongName, SetComments, which are defined in OriginObject class, can be used to control workbook's name (both short name and Long Name) and comments.

```
WorksheetPage wksPg("Book1");
if(wksPg)
{
        wksPg.SetName("MyBook");  // Rename workbook
        wksPg.SetLongName("This is Long Name", false);  // Set Long Name
        wksPg.SetComments("Comments");  // Set Comments
}
```

Also, Label property is provided for changing Long Name. And TitleShow property is for how to show short name and Long Name on the workbook's title.

```
WorksheetPage wksPg1("Book2");
if(wksPg1)
{
        wksPg1.Label = "My Label";  // Set Label (also called Long Name)
        // Show only Label on workbook's title
        wksPg1.TitleShow = WIN_TITLE_SHOW_LABEL;
        // Show only short name on workbook's title
        // wksPg1.TitleShow = WIN_TITLE_SHOW_NAME;
        // Show both short name and Label on workbook's title
        // wksPg1.TitleShow = WIN_TITLE_SHOW_BOTH;
}
```

6.2.3 Workbook Manipulation

Origin provides the capabilities for workbook manipulation by using Origin C, such as merging, splitting, etc.

6.2.3.1 Merge Workbooks

To merge many workbooks into one workbook, actually it is to copy the worksheets from the source workbooks to the target workbook. To add worksheet to a workbook, the AddLayer method is available.

The following example is to merge all workbooks in current folder to the newly created workbook.

```
WorksheetPage wksPgTarget;
wksPgTarget.Create("Origin");  // Create the target workbook
Folder fld = Project.ActiveFolder();  // Get the active/current folder
foreach(PageBase pb in fld.Pages)
{  // Loop all Pages in folder
        WorksheetPage wksPgSource = pb;  // Convert the Page to WorksheetPage
        // If convert failed, that is to say the Page is not WorksheetPage
        if(!wksPgSource)
        {
                continue;  // Next Page
        }
        // Skip the target workbook
        if(wksPgTarget.GetName() == wksPgSource.GetName())
        {
                continue;
        }
        // Loop all worksheet in workbook for merging
        foreach(Layer lay in wksPgSource.Layers)
        {
                Worksheet wks = lay;  // Get the worksheet
                // Add the worksheet to target workbook
                wksPgTarget.AddLayer(wks, 0, false);
        }
        // If not to keep the source workbook, destroy it
        wksPgSource.Destroy();
```

```
}
```

6.2.3.2 Split Workbook

The example above is merging multiple workbooks into one workbook. It is also able to split a workbook into multiple workbooks, which contain single worksheet.

```
WorksheetPage wksPgSource("Book1");  // Workbook with multiple worksheets

// Loop over all worksheets

foreach(Layer lay in wksPgSource.Layers)

{

        Worksheet wks = lay;  // Get worksheet

        WorksheetPage wksPgTarget;

        wksPgTarget.Create("Origin");  // Create new workbook

        wksPgTarget.AddLayer(wks);  // Add worksheet to the new workbook

        wksPgTarget.Layers(0).Destroy();  // Delete the first worksheet

}
```

6.3 Worksheet Columns

6.3.1 Worksheet 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 this sub-chapter.

This section covers the following topics:

Worksheet Column Operation

Worksheet Column Data Manipulation

6.3.2 Worksheet Column Operation

To perform operation on worksheet column, you can use Column class or Worksheet class.

6.3.2.1 Add or Insert Column

To add a column to the end of the worksheet, the AddCol method in Worksheet class is available, and also the InsertCol for inserting a column before a specified position.

```
// Add column with default name
int nColIndex = wks.AddCol();


// Add column with namestring 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



// 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);
}
```

6.3.2.2 Delete Column

The Worksheet::DeleteCol method is capable of removing a column from worksheet.

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

6.3.2.3 Rename and Label Column

To rename (short name) a column, Origin provides the SetName method.

```
Column col = wks.Columns(0);  // Get the 1st column in worksheet
BOOL bRet = col.SetName("MyNewName");  // Rename the column
```

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.

6.3.2.4 Hide/Unhide Column

To hide/unhide column(s), you can use the Workhseet::ShowCol method.

```
wks.ShowCol(1, 1, false); // to hide column 1.
```

6.3.2.5 Move and Swap Columns

Move Column To move columns or swap columns, the super class of Worksheet class, Datasheet class, provides the method MoveColumns and [[OriginC:Datasheet-SwapColumns|SwapColumns] respectively for such purposes.

```
// Move three columns - starting with column 5 - to the first column
// Example requires first worksheet in project with at least 7 columns
Worksheet wks = Project.ActiveLayer();
if(wks)
    wks.MoveColumns(4, 3, MOVE_COL_TO_FIRST);

// Reverse the column order in the active worksheet
for(int ii = 1; ii <= wks.GetNumCols() / 2 ; ii++)
    wks.SwapColumns(ii - 1, wks.GetNumCols() - ii);
```

6.3.2.6 Add Sparkline to Column

To add sparkline to column(s), Origin C provides the wks_set_show_labels with the RCLT_SPARKLINE label type.

```
```

```
// Configure active sheet to show Sampling Inverval and SparkLine in order

// append to the curernt Labels

Worksheet wks = Project.ActiveLayer();

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

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

6.3.2.7 Data Type, Format, SubFormat

6.3.2.7.1 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);
```

6.3.2.7.2 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);

}
```

6.3.2.7.3 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);

}
```

6.3.3 Worksheet Column Data Manipulation

6.3.3.1 Basic Arithmetic Operation

To perform the base arithmetic operation on the column data, you can first get the column data into vector, and then operate on the corresponding vectors.

```

// Get data from the 1st and 2nd columns

// Then add two columns together,

// and put results to 3rd column

Worksheet wks = Project.ActiveLayer();

if(!wks)

{

        return;
```

```
}

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

Column col2 = wks.Columns(1);  // 2nd column

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


vectorbase &v1 = col1.GetDataObject();  // Get data object

vectorbase &v2 = col2.GetDataObject();

vectorbase &v3 = col3.GetDataObject();

v3 = v1 + v2;  // Add together
```

6.3.3.2 Set 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();
```

6.3.3.3 Sort Column

To sort a specified column, first get the column's data into a vector, and then put the data back after sorting the vector. By using a vector reference for getting data object from column, the vector will attach to the column automatically, and the data update on vector will map back to column.

```


Worksheet wks = Project.ActiveLayer();

if(!wks)

{

        return;

}

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

vectorbase &v1 = col1.GetDataObject();  // Get data object using reference

v1.Sort(SORT_DESCENDING);  // Sort descendingly
```

6.3.3.4 Reverse Column

To reverse column's data, first you can get the column data into a vector, and then reverse the data in vector and put them back.

```

```

```
// Reverse the 1st column's data

Worksheet wks = Project.ActiveLayer();

if(!wks)

{

        return;

}

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

vectorbase &v1 = col1.GetDataObject();  // Get data object

vector<uint> vnIndices;  // vector for reverse indices

vnIndices.Data(v1.GetSize() - 1, 0, -1);  // Reverse indices

v1.Reorder(vnIndices);  // Reverse the data
```

6.3.3.5 Get & Set Data from Column

6.3.3.5.1 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]);
```

6.3.3.5.2 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);
```

6.3.3.5.3 Get & Set Date and Time Data from Column

If the column's format is Date or Time, the data you get from this column will be Julian date/time data, but not the display-date-time-format string.

```
// Get active worksheet
Worksheet wks = Project.ActiveLayer();
Column col1(wks, 0);  // The first column
Column col2(wks, 1);  // The second column
// Check if the first column's format is Date or Time, or not
if(col1.GetFormat() == OKCOLTYPE_DATE || col1.GetFormat() == OKCOLTYPE_TIME)
{
```

```
        // Get data from 1st column, v1 holds Julian data

        vector &v1 = col1.GetDataObject();

        vector &v2 = col2.GetDataObject();  // Get data from 2nd column

        v2 = v1;  // Set 1st column's Julian data to 2nd column

        col2.SetFormat(OKCOLTYPE_DATE);  // Set 2nd column to be Date column

        // Set display format to be MM/dd/yyyy HH:mm:ss

        col2.SetSubFormat(LDF_SHORT_AND_HHMMSS_SEPARCOLON);

}
```

6.3.3.6 Get the columns of different worksheets\workbooks

To calculate the sum of a specific column of all worksheets in all workbooks, you can loop all the worksheets in the current folder, and operate on the wanted column(s).

```
// Retrieve the second column of each worksheet in each workbook of current folder,
calculate its sum and output to a new worksheet.

void Calculate_Column_Sum()

{

    StringArray ColNames;

    vector<double> ColMeans;

    int K = 1;

    Dataset ds;

    double colSum;

    Folder fld = Project.ActiveFolder();  // Get the active/current folder

    foreach(PageBase pb in fld.Pages)

    {

            // Loop all Pages in folder

            WorksheetPage wksPgSource = pb;  // Convert the Page to WorksheetPage

            // If convert failed, that is to say the Page is not WorksheetPage

            if(!wksPgSource)
```

```
            {
                continue;  // Next Page
            }

            // Loop all worksheet in workbook
            foreach(Layer lay in wksPgSource.Layers)
            {
                Worksheet wks = lay;
                // Get column info
                ds.Attach(wks, K);
                DataRange dr;
                dr.Add("X", wks,0,K,-1,K);
                ColNames.Add(dr.GetDescription());
                // Calculate column sum
                ds.Sum(colSum);
                ColMeans.Add(colSum);
            }

    }
    // Prepare the resulting worksheet
    Worksheet wksResult;
    wksResult.Create("Origin");
    DataRange dr;
    dr.Add(wksResult,0,"X");
    dr.Add(wksResult,1,"Y");
    dr.SetData(&ColMeans,&ColNames);
}
```

6.4 Worksheets

6.4.1 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 Columns. The Worksheet class is derived from the Layer class.

This section covers the following topics:

Worksheet Basic Operation

Worksheet Data Manipulation

Converting Worksheet to Matrix

Virtual Matrix

6.4.2 Worksheet Basic Operation

The basic worksheet operations include adding worksheet to workbook, activating a worksheet, getting and setting worksheet properties, deleting worksheet, etc. Some practical examples are provided below.

6.4.2.1 Add New Worksheet

Add a worksheet to a workbook using the AddLayer method.

```
// 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);
```

6.4.2.2 Activate a Worksheet

Workbook is an Origin object that contains worksheets. To make a worksheet in workbook to be activated, the function set_active_layer can be used.

```
// Access a worksheet by full name
Worksheet wks("[Book1]Sheet1");

// Set this worksheet to be active
set_active_layer(wks);
```

6.4.2.3 Delete Worksheet

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
```

6.4.2.4 Access Worksheets in 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 = okutil_make_book_sheet_string(wksPage.GetName(), "Sheet1");

// If book and sheet name are known, the string can be constructed manually.
string strFullName = okutil_make_book_sheet_string("Book5", "Sheet1");
```

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
```

6.4.2.5 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");
```

6.4.2.6 Copy Worksheet

The 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
}
```

6.4.2.7 Format 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;

// Fill color
tr.Root.CommonStyle.Fill.FillColor.nVal = SYSCOLOR_LTCYAN;

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

// The font size of text
tr.Root.CommonStyle.Font.Size.nVal = 11;

// The color of text
tr.Root.CommonStyle.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);
}
```

6.4.2.8 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());
```

6.4.2.9 Read Only Cells

If you don't want the contents in a cell of worksheet to be changed, you can set the cells to be read-only by using theme tree.

The example below shows how to set the data cells in worksheet to be read-only, and then change the second data cell in column 1 to be editable.

```

```

```
// Create a worksheet by using default template (Origin)
// so to make sure that Long Name, Units, and Comments rows are shown
Worksheet wks;
wks.Create("Origin");

Tree tr;
tr = wks.GetFormat(FPB_ALL, FOB_ALL, true, true);  // Get theme tree of worksheet

// Start to get the specific tree node from the theme tree
// to set the read-only format for the data cells
string strName = "ogData";  // Use to get the node with the desired format
TreeNode trGrid, trNameStyles;
trGrid = tr.Root.Grid;  // Get Grid node
if(!trGrid.IsValid())
        return;

// Read-only format is under some child node of this node
trNameStyles = trGrid.NameStyles;
if(!trNameStyles.IsValid())
        return;

TreeNode trNameStyle;
bool bRet = false;
// Loop all children nodes to find out the desired tree node
foreach(trNameStyle in trNameStyles.Children)
{
        // Find the node with "ogData"
```

```
        if(0 == trNameStyle.Name.strVal.Compare(strName))
        {
                bRet = true;
                break;
        }
}
if(!bRet)
        return;

trNameStyle.Style.ReadOnly.nVal = 1;  // Set all data cells to be read-only

// Start to get/create the specific tree node from the theme tree
// to cancel the read-only format for the specified data cell
TreeNode trRangeStyles;
trRangeStyles = trGrid.RangeStyles;  // Get RangeStyles node from Grid node
TreeNode trRangeStyle;
if(!trRangeStyles.IsValid())  // If RangeStyles node does not exist yet
{
        // Create RangeStyles node
        trRangeStyles = trGrid.AddNode("RangeStyles");
        // And create a sub node named RangeStyle1
        trRangeStyle = trRangeStyles.AddNode("RangeStyle1");
}
else  // If RangeStyles node already exist
{
        // Find how many children nodes
        int tagNum = trRangeStyles.Children.Count();
```

```
        // And create a sub node name RangeStyle#, # = tagNum+1
        trRangeStyle = trRangeStyles.AddNode("RangeStyle"+(tagNum+1));
}
// Define the range for setting, here range is just one cell
// Left cell of the range, start from 1
trRangeStyle.Left.nVal = 1;
// Top cell of the range, start from 5, including label rows
// there are 4 label rows, then 5 means the first data cell
trRangeStyle.Top.nVal = 5;
// Just one cell, so right of the range is the same with left
trRangeStyle.Right.nVal = 1;
// Just one cell, so bottom of the range is the same with top
trRangeStyle.Bottom.nVal = 5;
trRangeStyle.Style.ReadOnly.nVal = 0;  // Set read-only to 0 to cancel it


// Apply the setting format to worksheet
if(0 == wks.UpdateThemeIDs(tr.Root))
{
        bool bb = wks.ApplyFormat(tr, true, true);
        if(bb)
        {
                printf("Cell 1 in column 1 is editable.\n");
        }
}
```

It is also able to set the Read-Only format for the cells in label rows. We can just make some simple changes on the code above. For example, we are going to make the Comments row to be read-only except the one in column 2, then the corresponding changes are like below.

```
/* Comment out the line below in the above code
string strName = "ogData";
*/
// This line is for the Data, just change it for Comments, as following
string strName = "ogComment";

/* Comment out the following 4 lines in the above code
trRangeStyle.Left.nVal = 1;
trRangeStyle.Top.nVal = 5;
trRangeStyle.Right.nVal = 1;
trRangeStyle.Bottom.nVal = 5;
*/
// These 4 lines are used to set for the second data
// cell (assume 3 label rows displayed in worksheet)
// Now we need to set for the Comments cell,
// assume the Comments row is the third row,
// and is for column 2, but not column 1 anymore, then
trRangeStyle.Left.nVal =
trRangeStyle.Right.nVal = 2;  // Column 2
// Comments row (the third row displayed in worksheet)
trRangeStyle.Top.nVal =
trRangeStyle.Bottom.nVal = 3;
```

6.4.3 Worksheet Data Manipulation

In this section we present examples of how to manipulate worksheet data by Origin C.

6.4.3.1 Get 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);
}
```

6.4.3.2 Set 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);
```

6.4.3.3 Put 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 forget to call this
}
```

6.4.3.4 Access 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
```

6.4.3.5 Sort 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);
```

6.4.3.6 Mask 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();
```

6.4.3.7 Set Size

The Worksheet::SetSize method is used to set the number of rows and columns in a worksheet.

```
// 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.
```

6.4.3.8 Reduce Worksheet Data

Origin C provides some functions for reducing XY data in worksheet, such as ocmath_reducexy_fixing_increbin for reducing XY data by X increment, ocmath_reducexy_n_groups for reducing XY data by number of groups, ocmath_reducexy_n_points for reducing XY data by every N points, etc. The following is an example to show how to reduce XY data by every N points.

```
```

```
Worksheet wks = Project.ActiveLayer();  // Get active worksheet
if(!wks)
{
        return;
}
Column colX(wks, 0);  // First column in worksheet
Column colY(wks, 1);  // Second column in worksheet
if(colX && colY)
{
        vectorbase &vbInterY = colY.GetDataObject();  // Get Y column data
        vector vY = vbInterY;
        vector vReduced(vY.GetSize());
        int nPoints = 3;
        // Reduce every 3 points, and result is the mean of every 3 points
        int nNewSize = ocmath_reducexy_n_points(vY, vReduced, vY.GetSize(),
                nPoints, REDUCE_XY_STATS_MEAN);
        int iReduced = wks.AddCol("Reduced");  // Add a new column for result
        Column colReduced(wks, iReduced);
        vectorbase &vbReduced = colReduced.GetDataObject();
        vbReduced = vReduced;
}
```

6.4.3.9 Extract 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);
}
```

6.4.3.10 Compare 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");
}
```

6.4.4 Converting Worksheet to Matrix

You may need to re-organize your data by converting from worksheet to matrix, or vice versa, for certain analysis or graphing needs. This page provides information and examples of converting worksheet to matrix, and please refer to Converting Matrix to Worksheet for the "vice versa" case.

6.4.4.1 Worksheet Gridding

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);
```

Include header files in the Origin C file.

```
#include <wks2mat.h>
#include <Nag_utils.h>
```

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);
```

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);
```

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;
}
```

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 matrixsheet
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
```

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");
}
```

6.4.4.2 Worksheet to Matrix

Data contained in a worksheet can be converted to a matrix using a set of functions.

To converts matrix-like worksheet data directly into a matrix, data in source worksheet can contain the X or Y coordinate values in the first column, first row. However, because the coordinates in a matrix should be uniform spaced, you should have uniformly spaced X/Y values in the source worksheet. The CopyFramWks method can be used directly, or just attach XYZ data range to matrix.

The following example show how to perform direct worksheet to matrix conversion:

```
// Method 1: using CopyFromWks
Worksheet wks = Project.ActiveLayer();
if(!wks)
{
```

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


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



// Method 2: attach to MatrixObject
Worksheet wks = Project.ActiveLayer();
if(!wks)
{
        return;
}
int nCols = wks.GetNumCols();
int nRows = wks.GetNumRows();
DataRange dr;
dr.Add("X", wks, 0, 1, 0, nCols - 1);  // First row excep the first cell
dr.Add("Y", wks, 1, 0, nRows - 1, 0);  // First column except the first cell
```

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

MatrixPage matPg;

matPg.Create("Origin");

MatrixLayer matLy = matPg.Layers(0);

MatrixObject mo = matLy.MatrixObjects(0);

MatrixObject moTmp;

moTmp.Attach(dr);

matrixbase &matTmp = moTmp.GetDataObject();

matrixbase &mat = mo.GetDataObject();

mat = matTmp;

moTmp.Detach();
```

When your worksheet data is organized in XYZ column form, you should use Gridding to convert such data into a matrix. Many gridding methods are available, which will interpolate your source data and generate a uniformly spaced array of values with the X and Y dimensions specified by you.

The following example converts XYZ worksheet data by Renka-Cline gridding method.

```

// Convert worksheet data into a 20 x 20 matrix by Renka-Cline gridding method

Worksheet wks = Project.ActiveLayer();

if(!wks)

{

        return;

}

Dataset dsX(wks, 0);

Dataset dsY(wks, 1);

Dataset dsZ(wks, 2);

int nPoints = dsX.GetSize();

vector vX = dsX;
```

```
vector vY = dsY;

vector vZ = dsZ;



ocmath_RenkaCline_Struct comm;

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




//set X and Y of the gridding

double dXMin, dXMax, dYMin, dYMax;

vX.GetMinMax(dXMin, dXMax);

vY.GetMinMax(dYMin, dYMax);



//perform random matrix conversion using Kriging algorithm

int nRows = 20;

int nCols = 20;

matrix mZ(nRows, nCols);

vector vEvalX(nRows * nCols);

vector vEvalY(nRows * nCols);

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

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

ocmath_renka_cline_struct_free(&comm);



//create Matrix storing the result

MatrixLayer    mResultLayer;

mResultLayer.Create();

Matrix    matResult(mResultLayer);
```

```
matResult = mZ;

MatrixObject mo = mResultLayer.MatrixObjects(0);

mo.SetXY(dXMin, dYMin, dXMax, dYMax);//set X and Y range of Matrix
```

6.4.5 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);
```

7 Graphs

7.1 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
```

This section covers the following topics:

Creating and Customizing Graph

Adding Data Plots

Customizing Data Plots

Managing Layers

Creating and Accessing Graphical Objects

7.2 Creating and Customizing Graph

7.2.1 Creating Graph Window

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

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

7.2.2 Getting Graph Page Format

```
GraphPage gp("Graph1");

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

7.2.3 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);
```

7.2.4 Getting Graph Layer Format

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

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

7.2.5 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);
```

7.2.6 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
```

7.2.7 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);
}
```

7.2.8 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.nVal = 0; // 0=increment by value; 1=number of major ticks
axesX.Scale.Value.dVal = 0.2; // Increment value
axesX.Scale.Type.nVal = 0;// Linear
axesX.Scale.Rescale.nVal = 0; // Rescake type
axesX.Scale.RescaleMargin.dVal = 8; // precent 8
```

This example shows how to set scale major ticks number for Y axis.

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

Axis axesY = gl.YAxis;



axesY.Scale.IncrementBy.nVal = 1; // 0: increment by value; 1: number of major
ticks

axesY.Scale.MajorTicksCount.nVal = 5;
```

7.2.9 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);
```

7.2.10 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();
```

7.2.11 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);
}
```

7.2.12 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);
```

7.2.13 Customizing 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);
}
```

7.2.14 Change Scale Factor

This example shows how to scale the font size and line thickness when page size changed by changing scale factor.

```
void change_graph_and_font_size(double dNewWidth = 5)
{
        GraphPage gp = Project.ActiveLayer().GetPage();
        Tree tr1;tr1 = gp.GetFormat(FPB_ALL, FOB_ALL, true, true);
        double dOldWidth = tr1.Root.Dimension.Width.dVal;
        double factor = dOldWidth/tr1.Root.Dimension.Height.dVal;
        Tree tr2;
        tr2.Root.Dimension.Width.dVal = dNewWidth;
        tr2.Root.Dimension.Height.dVal = dNewWidth / factor;
        if(0 == gp.UpdateThemeIDs(tr2.Root))
        {
                gp.ApplyFormat(tr2, true, true);
                string strScript;
```

```
                //page -afu : Change Scale Factor

                //win -z0 : fit page to window size

                strScript.Format("page -AFU %f;win -z0", -dNewWidth/dOldWidth);

                gp.LT_execute(strScript);

        }
}
```

Note:

In this case, tr1 is only used to get current dimension value and does not apply new value.

It's because tr1 also has graph object position settings, which are outdated if you want to change graph dimension.

The correct way is to apply dimension setting only and graph object position will automatically update.

7.3 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.

7.3.1 2D Plot (XY, YErr, Bar/Column)

7.3.1.1 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

}
```

7.3.1.2 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);
```

7.3.1.3 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();

}
```

7.3.2 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");
}
```

7.3.3 Contour Plot

7.3.3.1 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");
}
```

7.3.3.2 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();
}
```

7.3.4 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();
}
```

7.3.5 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);
```

7.3.6 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.nVal
=

gl.YAxis.AxisObjects(AXISOBJPOS_LABEL_SECOND).RightLabels.Color.nVal =
                INDEX_COLOR_AUTOMATIC;


                gl.Rescale();
        }
}
```

7.4 Customizing Data Plots

7.4.1 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();

}
```

7.4.2 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);
```

7.4.3 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.
```

7.4.4 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);
}
```

7.4.5 Copying Format from One Data Plot to Another

7.4.5.1 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);
```

7.4.5.2 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);
```

7.4.6 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);
}
```

7.4.7 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

}
```

7.4.8 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);
```

7.5 Managing Layers

7.5.1 Creating a Panel Plot

7.5.1.1 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);
```

7.5.1.2 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++;
}
```

7.5.2 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);
```

7.5.3 Hiding Layers Except Active One

```
GraphPage gp("Graph1");

if( gp )

{

        GraphLayer glActive = gp.Layers(-1); // -1 to get active layer



        foreach(GraphLayer gl in gp.Layers)

        {

                if( gl.GetIndex() != glActive.GetIndex() )

                        gl.Show(false);

        }

}
```

7.5.4 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);
```

7.5.5 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);
```

7.5.6 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);
```

7.5.7 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);
```

7.5.8 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);
```

7.5.9 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);

}
```

7.5.10 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);
```

7.6 Creating and Accessing Graphical Objects

7.6.1 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);
```

```
}
```

7.6.2 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);
}
```

7.6.3 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);

}
```

7.6.4 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;
```

7.6.5 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");
}
```

7.6.6 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);

}
```

7.6.7 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
```

7.6.8 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);
```

8 Working with Data

8.1 Working with Data

This section covers the following topics:

Numeric Data

String Data

Date and Time Data

8.2 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:

double

integer

vector

matrix

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

Note:Values such as 0.0, NANUM (missing value) and values between -1.0E-290 to 1.0E-290 will be evaluated to be *False* in logic statement.

8.2.1 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.");
```

8.2.2 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 3.141592653589793238462643

// 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");
```

```
}
```

8.2.3 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);
```

8.2.4 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
```

8.2.5 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
```

8.2.6 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
```

8.2.7 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;
}
```

8.2.8 DataRange

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

8.2.8.1 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);
```

8.2.8.2 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);
```

8.2.8.3 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);
```

8.2.8.4 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

}
```

8.3 String Data

8.3.1 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.";
```

8.3.2 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
```

8.3.3 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);
```

8.3.4 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"
```

8.3.5 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);
```

8.3.6 Path String Functions

8.3.6.1 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);
```

8.3.6.2 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);
```

8.4 Date and Time Data

Origin C provides support for date and time data.

8.4.1 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);
```

8.4.2 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);
```

8.4.3 Convert String to Julian Date

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

9 Projects

9.1 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.

This section covers the following topics:

Managing Projects

Managing Folders

Accessing Pages

Accessing Metadata

Accessing Operations

9.2 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.

9.2.1 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
```

9.2.2 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);
```

9.2.3 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();
}
```

9.3 Managing Folders

Pages in an Origin project (workbooks, matrixbooks, 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.

9.3.1 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());
```

9.3.2 Get the Active Folder

```
Folder fldActive;
fldActive = Project.ActiveFolder();

// Add a sub folder to it
Folder fldSub;
fldSub = fldActive.AddSubfolder("MyFolder");
printf("Folder added successfully, path is %s\n", fldSub.GetPath());
```

9.3.3 Activate a Folder

```
// activate root folder
Folder fldRoot = Project.RootFolder;
fldRoot.Activate();

// activate the specified sub folder
Folder fldSub("/MyFolder");
fldSub.Activate();
```

9.3.4 Get Path for a Specific Page

```
GraphPage gp("Graph1");
if( gp.IsValid() )
```

```
{
        Folder fld = gp.GetFolder();
        out_str(fld.GetPath());
}
```

9.3.5 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 = Project.RootFolder.AddSubfolder("sub1");
Folder subfld2 = Project.RootFolder.AddSubfolder("sub2");

// Move the sub2 folder under the sub1 folder
if( !Project.RootFolder.Move(subfld2.GetName(), "/"+subfld1.GetName()+"/", true) )
    printf("move folder failed!");
```

9.4 Accessing Pages

Pages in Origin consist of workbooks, matrixbooks 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.

9.4.1 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
```

9.4.2 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 matrixsheet.

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();
```

9.4.3 Activate One Page

If want to activate a window, you can use PageBase::SetShow(PAGE_ACTIVATE) to cause the window to be activated.

```
// attach to a graph window named Graph2
GraphPage gp( "Graph2" );

// set the window to be active
gp.SetShow( PAGE_ACTIVATE );
```

9.4.4 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( PageBase pg in Project.Pages )
{
    out_str(pg.GetName());
}
```

9.5 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.

9.5.1 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.

9.5.2 Access Tree

9.5.2.1 Access a Tree in a Project

9.5.2.1.1 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);
```

9.5.2.1.2 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);
```

9.5.2.1.3 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);
```

9.5.2.2 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.

9.5.2.2.1 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);
}
```

9.5.2.2.2 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

}
```

9.5.2.2.3 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]);
}
```

9.5.2.3 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.

9.5.2.3.1 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);
```

9.5.2.3.2 Get Tree

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

```
Tree tr;
if( col.GetBinaryStorage("colTree", tr) )
        out_tree(tr);
```

9.5.2.3.3 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]);
```

9.5.2.4 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);
```

9.5.2.5 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);

}
```

9.6 Accessing Operations

9.6.1 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);
}
```

9.6.2 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");
```

9.6.3 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);
```

10 Importing

10.1 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, video files, and data from a database. The following sections will show you how to import data into a worksheet or matrix sheet.

This section covers the following topics:

Importing Data

Importing Images

Importing Videos

10.2 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.

10.2.1 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.
    }
}
```

10.2.2 Import ASCII Data File into Matrixsheet

Importing data into a matrixsheet 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 matrixsheet from the active matrixbook 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.");
}
```

10.2.3 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");
```

```
}
```

10.2.4 Import Files with Import Wizard

There are times when the data files are neither ASCII nor simple binary files or there is no existing filter for importing a data file, in these cases you can use Origin C and impFile X-Functions to import the files with the Import Wizard.

The **Origin C** function should have either of the following prototypes:

int YourFunctionName(Page& pgTarget, TreeNode& trFilter, LPCSTR lpcszFile, int nFile)

where:

pgTarget: A reference to a Page object of type worksheet or Matrix. This would be what you defined in your filter or on the Source page of the Import Wizard, as the target window.

trFilter: A reference to a TreeNode object that holds all the filter settings from your filter file, or from your wizard specifications, in a tree structure.

lpcszFile: The full path and name of the file that is being imported.

nFile: The file index number in an ordered sequence of imported files (e.g. If you import n files, your function gets called n times, and nFile is the file count for the file being processed).

Or

int YourFunctionName(Layer& lyTarget, TreeNode& trFilter, LPCSTR lpcszFile, int nFile)

where:

lyTarget: A reference to a Layer object of type worksheet or Matrix. This would be what you defined in your filter or on the Source page of the Import Wizard, as the target window.

trFilter: A reference to a TreeNode object that holds all the filter settings from your filter file, or from your wizard specifications, in a tree structure.

lpcszFile: The full path and name of the file that is being imported.

nFile: The file index number in an ordered sequence of imported files (e.g. If you import n files, your function gets called n times, and nFile is the file count for the file being processed).

See an example in the \Samples\Import and Export\User Defined folder, found in the Origin installation folder.

Note: The target window template named on the first page of the Import Wizard (**Source** page) is only used when creating new windows (as would happen under some conditions during drag-and-drop importing). When choosing **File: Import**, if your active window is consistent with your import filter's **Target**

Window specification, no new window is created and a reference to the page object for the active window is passed to your function. If the active window is of a different type, a new window is created using the specified template, and the page reference to this new window is passed.

10.2.4.1 Variable Extraction in Import Wizard

When importing ASCII files with the Import Wizard, you can extract variables from the file headers using user-defined Origin C functions.

Your custom Origin C function should have the following prototype:

int FuncName(StringArray& saVarNames, StringArray& saVarValues, const StringArray& saHdrLines, const TreeNode &trFilter);

where:

saVarNames: An string array where the user should put the variable names.

saVarValues: An string array where the user should put the variable values.

saHdrLines: A reference to an string array that contains the header lines. Note that the Origin C function does not need to read the data file because the header lines are automatically passed into the function.

trFilter: A reference to a TreeNode object that holds all the filter settings from your filter file, or from your wizard specifications, in a tree structure.

10.3 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.

10.3.1 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, // matrixbook 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;
}
```

10.3.2 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();
}
```

10.3.3 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);
```

10.4 Importing Videos

10.4.1 Version Info

Minimum Origin Version Required: Origin 93 SR0

Origin C provides the VideoReader class for reading a video file and importing frame(s) of video to matrix object(s).

To use the VideoReader class, the header file "VideoReader.h" needs to be included in your source code.

```
#include <..\OriginLab\VideoReader.h>
```

With the VideoReader class you can open a video file and get the video's properties, such as frame count, frame rate (frame per second), current position, etc. It also provides methods for seeking frame, seeking time, and reading frame(s) into matrix object(s).

The following example will create a new matrixbook, seek 10 frames into a video then load 100 frames into 100 matrix objects in the active matrixsheet by skipping every other frame.

```
#include <..\Originlab\VideoReader.h>  // Include the header file
```

```
void Import_Video_Ex1(string strFile = "d:\\test.avi") {
        MatrixLayer ml;
        ml.Create("Origin");  // Create a matrixsheet for the frames
        char str[MAXLINE];
        VideoReader vr;  // Declare a VideoReader
        strcpy(str, strFile);
        if(!vr.Open(str)) {  // Open the video file
                out_str("Failed to open video file!");
                return;
        }
        // Get number of frames
        int iFrameCount = (int)vr.GetFrameCount();
        printf("%u frames\n",iFrameCount);
        // Starting frame
        int iOffset = 10;
        // Specify total frames to read
        int iTotalFrames = 100;
        // Specify frames to skip between each read
        int iSkip = 1; // Read every other frame
        bool bRet = vr.SeekFrame(iOffset);
        vr.ReadFrames(ml, iTotalFrames, iSkip);  // Read frames
        if(vr.ReaderReady()) {
                vr.Close();  // Close the video reader
        }
}
```

In this example, time is used as the metric by which we seek and import with time skips..

```
#include <..\Originlab\VideoReader.h>  // Include the header file

void Import_Video_Ex2(string strFile = "d:\\test.avi") {

        MatrixLayer ml;

        ml.Create("Origin");  // Create a matrixsheet for the frames

        char str[MAXLINE];

        VideoReader vr;  // Declare a VideoReader

        strcpy(str, strFile);

        if(!vr.Open(str)) {  // Open the video file

                out_str("Failed to open video file!");

                return;

        }

        // Get number of frames

        int iFrameCount = (int)vr.GetFrameCount();

        // Get frame rate

        double dFPS = vr.GetFPS();

        double dRunningTime = iFrameCount / dFPS;

        printf("%u frames at %f fps with a running time of %f seconds\n",

                iFrameCount, dFPS, dRunningTime);


        // Setup for read

        double dStartTime = 5; // Begin reading at 5 seconds

        double dSkipLength = 3.333; // Skip 3.333 seconds between reads

        vr.SeekFrame((int) dStartTime * dFPS); // Calculate frame start

        int iSkip = (int) dSkipLength * dFPS; // Calculate frames to skip

        // Calculate number of frames to actually read

        int iTotalFrames = (int) ( (dRunningTime - dStartTime) * dFPS)

        / (iSkip + 1);
```

```
        vr.ReadFrames(ml, iTotalFrames, iSkip);  // Read frames

        if(vr.ReaderReady()) {

                vr.Close();  // Close the video reader

        }
}
```

11 Exporting

11.1 Exporting

This section covers the following topics:

Exporting Worksheets

Exporting Graphs

Exporting Matrices

Exporting Videos

11.2 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
```

11.3 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, 8);
}
```

11.4 Exporting Matrices

An Origin Matrix can be exported to an ASCII data file or an image file.

11.4.1 Export 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 add *#include <oExtFile.h>* 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);
```

11.4.2 Export Image from Matrix to Image File

The following example shows how to export a matrix to an image file.

Prior to running the following example, the image_utils.c file need to be loaded and compiled. This can be done from script with the following command or just add this file to your workspace.

```
run.LoadOC(Originlab\image_utils.c);
```

And need to add *#include <image_utils.h>* for the **export_Matrix_to_image** function.

```
MatrixLayer ml = Project.ActiveLayer();
MatrixObject mo = ml.MatrixObjects();
export_Matrix_to_image("c:\\matrixImg.jpg", "jpg", mo);
```

11.5 Exporting Videos

11.5.1 Version Info

Minimum Origin Version Required: Origin 9 SR0

Origin allows user to create a video with a collection of graphs. Origin C allows access to this ability using the Video Writer, you can define the video codec for compression (Please refer to FourCC Table for more details.), create a video writer project specifying the video name, path, speed and dimension, write graph pages as frames.

Note: To use the video writer, you must include its head file:

```
#include <..\OriginLab\VideoWriter.h>
```

The following example will write each graph in the project as a frame into the video, and the video is uncompressed with 2 frames/second speed and 800px * 600 px dimension.

```
    // Use the raw format without compression.
```

```
int codec = CV_FOURCC(0,0,0,0);


// Create a VideoWriter object.
VideoWriter vw;
int err = vw.Create("D:\\example.avi", codec, 2, 800, 600);
if (0 == err)
{
  foreach(GraphPage grPg in Project.GraphPages)


  // write the graph page into the video
  err = vw.WriteFrame(grPg);
}


// Release the video object when finished.
vw.Release();


return err;
```

The following example shows how to individually write a graph page into video and define the number of frames of this graph page in the video.

```
GraphPage gp("Graph1");


// The defined graph page will last for 10 frames.
int nNumFrames = 10;


vw.WriteFrame(gp, nNumFrames);
```

12 Analysis and Applications

12.1 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.

This section covers the following topics:

Mathematics

Statistics

Curve Fitting

Signal Processing

Peaks and Baseline

Using NAG Functions

12.2 Mathematics

12.2.1 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];
        }
    }
}
```

12.2.2 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);
```

12.2.3 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_nag.h> // NAG declarations
```

12.2.3.1 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,Nag_User *comm)
{
        int *use_comm = (int *)comm->p;
        return (x*sin(x*30.0)/sqrt(1.0-x*x/(PI*PI*4.0)));
}


void nag_d01sjc_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;

        Nag_User comm;
        static int use_comm[1] = {1};
        comm.p = (Pointer)&use_comm;
        d01sjc(func, a, b, epsabs, epsrel, max_num_subint, &result, &abserr,
        &qp, &comm, &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);
}
```

12.2.3.2 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);
}
```

12.2.3.3 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);
        }
}
```

12.2.4 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");
```

12.3 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.

12.3.1 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);
```

12.3.2 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.

12.3.3 Correlation 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");

}
```

12.3.4 Normality 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);

}
```

12.4 Curve Fitting

12.4.1 Curve Fitting

This section covers the following topics:

Linear Fitting

Polynomial Fitting

Multiple Regression

Non-linear Fitting

Find XY

12.4.2 Linear 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.

12.4.2.1 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.

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()

{

}
```

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
```

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;
}
```

12.4.2.2 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
}
```

12.4.2.3 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.

12.4.2.3.1 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();
```

```
}
```

12.4.2.3.2 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();
}
```

12.4.3 Polynomial 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.

12.4.3.1 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;
}
```

12.4.3.2 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.

12.4.4 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.

12.4.4.1 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;
}
```

12.4.4.2 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.

12.4.5 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.

12.4.5.1 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);
```

12.4.5.2 Accessing FDF File

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);

}
```

12.4.6 Find 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.

12.4.6.1 Linear

The formula of getting Y from X:

```
y = a + x * b;
```

The formula of getting X from Y:

```
x = (y - a) / b;
```

12.4.6.2 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.

Prior to running the following example, the nlsf_utils.c file need to be loaded and compiled. This can be done from script with the command run.LoadOC(Originlab\nlsf_utils.c) or just add this file to your workspace.

12.4.6.2.1 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);
}
```

12.4.6.2.2 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]);
}
```

12.5 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.

12.5.1 Smoothing

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);
```

12.5.2 FFT

Before using fft_* functions, you need to include fft_utils.h.

```
#include <fft_utils.h>
```

12.5.2.1 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
```

12.5.2.2 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
```

12.5.2.3 IFFT

```
fft_real(vSig.GetSize(), vSig, FFT_BACKWARD); // return 0 for no error
```

12.5.2.4 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");
}
```

12.5.3 FFT 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);
```

12.5.4 Wavelet 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);
```

12.6 Peaks and Baseline

12.6.1 Getting input XY from Graph or Worksheet

For the input XY data the following sections mentioned, we can get from Graph or Worksheet. Please click here to get the help of getting data from window.

12.6.2 Creating 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);
}
```

12.6.3 Removing 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;
```

12.6.4 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]);
}
```

12.6.5 Integrating and Fitting Peaks

12.6.5.1 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);
}
```

12.6.5.2 Fitting Peak

The Origin C **NLFitSession** class supports a method to fit peaks with a different fitting function.

12.7 Using NAG Functions

12.7.1 Header 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_nag.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 <NAG\nag.h>    // NAG struct and type definitions
#include <NAG\nagf02.h> // contains the f02 function declarations
```

12.7.2 Error 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);
```

12.7.3 Callback 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.

12.7.3.1 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);
```

12.7.4 NAG Get Data From Origin

Many NAG functions take a pointer to an array of numeric data. Both Origin worksheets and matrixsheets 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.

12.7.4.1 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);
```

12.7.4.2 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;
}
```

12.7.5 How to Call NAG e04 Functions

The following example will show how to call the NAG function, **nag_opt_simplex_easy**, safely. And the results will output to a file.

```
#include <OC_nag.h>

#define NULLFN NULL

void text_e04cbc()
{
        double objf;
        double x[2];
        Integer maxcal, n;
        NagError fail;
```

```
        printf("\ne04cbc example: \n");


        maxcal = 100;

        n = 2;

        x[0] = 0.4;

        x[1] = -0.8;

        tolf = sqrt(nag_machine_precision);

        tolx = sqrt(tolf);

        try

        {

                // call the NAG function, e04cbc = nag_opt_simplex_easy

                nag_opt_simplex_easy(n, x, &objf, tolf, tolx, funct, NULLFN,
maxcal, 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


        printf("The final function value is %12.4f\n", objf);

        printf("at the point");

        for (int ii = 1; ii <= n; ++ii)

        {

                printf(" %12.4f", x[ii-1]);

        }
```

```
        printf("\n");
}


// call back function for nag_opt_simplex_easy
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);
}
```

13 Output Objects

13.1 Output Objects

This section covers the following topics:

Results Log

Script Window

Notes Window

Report Sheet

13.2 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);
```

13.3 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
```

13.4 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");
```

13.5 Report Sheet

The Datasheet class has the **GetReportTree** and **SetReportTree** methods for getting and setting a report into a worksheet or matrixsheet.

```
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.");
```

14 Accessing Database

14.1 Accessing Database

This section covers the following topics:

Importing from a Database

Exporting into a Database

Accessing SQLite Database

14.2 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;
```

14.3 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.

Set up a database named "Analysis" in MySQL, and assume it is running on the machine "Lintilla".

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.

Open OriginExe\Samples\Curve Fitting\autofit.ogw, and fill the columns on the "Data" layer with data.

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;PWD=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);

}
```

14.4 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 DLLs for accessing 32-bit and 64-bit 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);
}
```

Origin C supports put database query result to vector:

```
#include <..\Originlab\oSQLite.h>

#define    STR_DATABASE_FILE       "E:\\DataSet1.1.db"

#define    STR_QUERY_STRING        "select * from Originlab limit 80"

void test_OSQLite_ex2()

{

        OSQLite sqlObj(STR_DATABASE_FILE);

        LPCSTR lpSQL = STR_QUERY_STRING;

        sqlObj.Select(lpSQL);

        vector<string> vsData;

        vector<double> vdData;

        sqlObj.GetFieldDataByName(&vsData, &vdData, "ID");

}
```

15 Accessing LabTalk

15.1 Accessing LabTalk

This section covers the following topics:

Getting and Setting Values for LabTalk Variables

Running LabTalk Script

Embedding LabTalk Script in Origin C Code

15.2 Getting and Setting Values for LabTalk Variables

Origin C has the ability to get and set LabTalk numeric and string values and run LabTalk scripts.

15.2.1 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. Class member functions **OriginObject::GetProp** and **OriginObject::SetProp** can get and set 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

    //

}
```

This is how to set and get line object color.

```

//This example assumes the active graph has a line object

void SetProp_Ex1(int nColor = 0, string strName = "line")

{

    GraphLayer    gl = Project.ActiveLayer();

    GraphObject    grobj;

    grobj = gl.GraphObjects(strName);

    double vv;

    if(grobj)

    {
```

```
        int nn = grobj.SetProp("color", nColor);
        out_int("SetProp return: ", nn);
        out_int("new color: ", grobj.GetProp("color"));
    }
    else
        out_str("no such obj");
}
```

15.2.2 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. Class member functions **OriginObject::GetProp** and **OriginObject::SetProp** can get and set 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");
```

15.3 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
```

You can also use *OriginObject::SetProp* function to change the LabTalk object properties. You can find more in the See_Also section.

```


wks.SetProp("colWidth", 8);
```

15.4 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);
```

16 Accessing X-Function

Origin has many built-in X-Functions for handling a variety of tasks. X-Functions can be called from both LabTalk and Origin C. This Section will show you how to call X-Functions from Origin C by using Origin C's XFBase class. This mechanism also can be used in calling one X-Function in another X-Function.

The XFBase class is declared in the XFBase.h header file located in the Origin C System folder. The XFBase.h header file is not included in the Origin.h header file so it has to be included separately in any Origin C file for the use of XFBase class.

```
#include <XFBase.h>
```

16.1 Calling the impFile X-Function From Origin C

The procedure to use X-Functions in Origin C is as following:

Declare an object that is to be constructed from a specified X-Function

Assign arguments using the SetArg options from the X-Function object

Execute the X-Function using the Evaluate method of the X-Function object

The following Origin C code defines a general function for importing files into Origin. The function has two arguments: data file name and import filter file name. Additional arguments of the **impFile** X-Function will use their default values.

```
bool call_impFile_XF(LPCSTR lpcszDataFile, LPCSTR lpcszFilterFile)
{
    string strDataFile = lpcszDataFile;
    string strFilterFile = lpcszFilterFile;

    // Create an instance of XFBase using the X-Function name.
    XFBase xf("impFile");
    if (!xf)
        return false;
```

```
    // Set the 'fname' argument.
    if (!xf.SetArg("fname", strDataFile))
        return false;

    // Set the 'filtername' argument.
    if (!xf.SetArg("filtername", strFilterFile))
        return false;

    // Call XFBase's 'Evaluate' method to execute the X-Function
    if (!xf.Evaluate())
        return false;

    return true;
}
```

The following Origin C code shows how to call the call_impFile_XF function defined above and use it to import an image file.

```
// Import the Car bitmap located in Origin's Samples folder.
string strImageFile = GetAppPath(TRUE) +
    "Samples\\Image Processing and Analysis\\Car.bmp";

// Import the bitmap using the Image import filter.
string strFilterFile = GetAppPath(TRUE) + "Filters\\Image.oif";

// Call the function that will use the impFile X-Function.
call_impFile_XF(strImageFile, strFilterFile);
```

Note:

For more example on accessing X-Function, see **Help: Programming > Origin C > Examples > Accessing X-Functions > Accessing X-Function**

For more information on X-Function, see **Help: X-Functions**

17 Calling Python Functions from Origin C

Python functions can be called from Origin C using the syntax

```
Python.function_name(arg1, arg2...)
```

By default the **.py** file should be placed in the same folder as the C/CPP or XF file. The name of the file should be **origin.py**.

Example - Python Function to perform Bayesian regression:

```
import numpy as np

import pandas as pd

import originpro as op

from sklearn import linear_model



def bayesian(vX, vY):

    blr = linear_model.BayesianRidge(tol=1e-6, fit_intercept=True,
compute_score=False, alpha_init=1, lambda_init=1e-3)

    blr.fit(np.vander(vX, 10), vY)

    mean = blr.predict(np.vander(vX, 10))

    return mean
```

The function can then be called from OC code:

```
void bayesian_reg()

{

        // Have a worksheet active, with three columns

        // Columns 1 and 2 have the X and Y data for fitting

        // Column 3 will have the fit curve result
```

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

        Dataset dsx, dsy, dsypred;

        dsx.Attach(wks, 0);

        dsy.Attach(wks, 1);

        dsypred.Attach(wks, 2);



        dsypred = Python.bayesian(dsx, dsy);
}
```

Multiple variables can be returned from Python function through arguments. **&** sign before the argument name indicates it's a returned variable.

```
Python.function_name(arg1, arg2..., & ret1, & ret2...)
```

Example - Python Function to perform Bayesian regression to return both fitted values and fitted errors:

```
import numpy as np

import pandas as pd

import originpro as op

from sklearn import linear_model



def bayesian(vX, vY, vYmean, vYstd):

    blr = linear_model.BayesianRidge(tol=1e-6, fit_intercept=True,
compute_score=False, alpha_init=1, lambda_init=1e-3)

    blr.fit(np.vander(vX, 10), vY)

    mean, std = blr.predict(np.vander(vX, 10), return_std=True)

    vYmean[:] = mean # pass mean values to variable vYmean

    vYstd[:] = std  # pass error values to variable vYstd
```

The function can then be called from OC code:

```
void bayesian_reg()
{
        // Have a worksheet active, with four columns
        // Columns 1 and 2 have the X and Y data for fitting
        // Column 3 and column 4 will have the fit curve result, and fitted error.
        Worksheet wks=Project.ActiveLayer();
        Dataset dsx, dsy, dsypred, dsystd;
        dsx.Attach(wks, 0);
        dsy.Attach(wks, 1);
        dsypred.Attach(wks, 2);
        dsystd.Attach(wks, 3);
        vector vYmean, vYstd;
        Python.bayesian(dsx, dsy, &vYmean, &vYstd);
        dsypred = vYmean;
        dsystd = vYstd;
}
```

To change the file location, you can set the LabTalk Python object property: **Python.OCWD$** to the full path of the desired folder (the working directory).

To change the file name, you can set the LabTalk Python object property: **Python.OCWF$** to the name of the desired Python file in the working directory.

18 User Interface

18.1 User Interface

This chapter demonstrates ways in which Origin C functions allow user interaction.

This section covers the following topics:

Dialog

Wait Cursors

Picking Points from a Graph

Adding Controls to a Graph

18.2 Dialog

18.2.1 Dialog

A graphic user interface(GUI) is one important part of your customized application in Origin. Creating GUI is associated with generating dialog in the Origin environment. This section demonstrates several ways to bring up dialog in Origin:

The simplest dialog can be generated using the **Built-in dialog boxes** function in Origin C (OC). the method supports several basic functionalities such as showing messages, opening files, saving files, etc.

If you need more controls of the dialog elements and also have some experience with OC, **GetN** Macros and **GetNBox** function in OC would be the right choice. More or less like C++, this method grants the access to each element in dialog to which you can add your own application script. **GetN** dialog provides an easier way to access and output data in Origin's worksheet.

A more automatic way to build dialog is probably using **X-function builder**. With this method, Origin generates dialog automatically for you and your task is merely defining the inputs and filling out spaces where your application logic is required. **X-Function** dialog can support **Recalculate** feature.

Python is embedded in Origin since Origin 2015. By calling Python in Origin, interactive dialog can also be generated with Python modules such as **tkinter**. It is very easy to learn how to build a **Python** dialog.

An even more sophisticated way would be to use Microsoft Visual C++ generated resource DLL for building floating tools, dialog boxes, and wizards in Origin. All elements in Origin, e.g. windows, worksheets, graphs can be accessed and controlled by this method. Using **Visual C++ DLL** dialog, you can build a very complex GUI.

Lastly, starting from Origin 2017, we add support for dialogs using HTML and JavaScript which adds more flexibility to making dialog that interacts with Origin. There are hundreds of third party libraries to call for **HTML** dialog. Using **HTML** dialog, you can build a very nice and complex GUI.

This section covers the following topics:

Built-in Dialog Boxes

GetN Dialog

X-Function

Dialog Builder

Origin C HTML Dialog with JavaScript Support

18.2.2 Built-in Dialog Boxes

18.2.2.1 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);
```

18.2.2.2 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.");
```

The next example shows a private reminder message dialog. An Ini file is used to initialize the dialog. Each section in the ini file is used for a single message.

```
```

```
/* Example Dialog.ini file in UFF.

[MyMessage]

;Title = My Reminder

Msg = This is my message.

;Btns = 4 for Yes No buttons

Btns = 4

*/

void PrivateReminderMessage_ex1()

{

   string iniFileName = GetOriginPath(ORIGIN_PATH_USER) + "Dialog.ini";

   int nRet = PrivateReminderMessage("MyMessage", iniFileName);

   printf("User chose %d\n", nRet);

}
```

18.2.2.3 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");
}
```

18.2.2.4 File 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.

18.2.2.4.1 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);
```

18.2.2.4.2 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));
```

18.2.2.4.3 File SaveAs Dialog

```
string strDefaultFilename = "Origin";

FDLogUseGroup nFDLogUseGroup = FDLOG_ASCII; // ASCII file group


string strPath = GetSaveAsBox(nFDLogUseGroup,GetAppPath(false),strDefaultFilename);

out_str( strPath );
```

18.2.2.4.4 Path Browser Dialog

```
string strPath = BrowseGetPath(GetAppPath() + "OriginC\\", "This is an example");

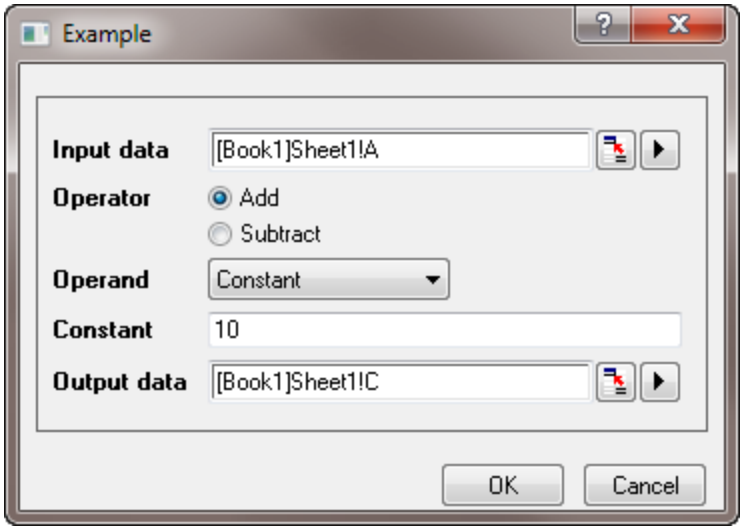
out_str(strPath);
```

18.2.3 GetN Dialog

18.2.3.1 Simple Dialog

There is an easy way to create a simple dialog using GetN macros and GetNBox function.

The dialog looks below:



Run the following function to open the upper dialog:

```
#include <GetNbox.h>
void simple_dialog()
{
    GETN_BOX(trRoot)    // define a Tree variable named "trRoot"

    // data range control
    GETN_INTERACTIVE(Input, "Input data", "[Book1]Sheet1!A")

    // radio button
    GETN_RADIO_INDEX_EX(Operator, "Operator", 0, "Add|Subtract")

    // list box
    GETN_LIST(type, "Operand", 0, "Constant|Reference Data")

    // string edit box
    GETN_STR(Constant, "Constant", "10")

    // choose a data range for reference data
    GETN_INTERACTIVE(Reference, "Reference Data", "[Book1]Sheet1!B")

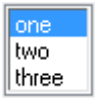
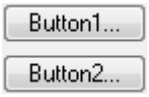
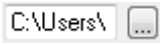
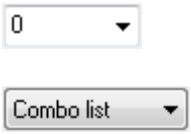
    // choose a column for output data
    GETN_INTERACTIVE(Output, "Output data", "[Book1]Sheet1!C")

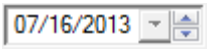
    // bring up dialog
    GetNBox(trRoot );
```

```
}
```

18.2.3.2 Controls

The following table lists the commonly used controls. For more controls and style setup, please refer to Origin C Reference: Macros: GetN.

Picture	Name
	GETN_RADIO_INDEX GETN_RADIO_INDEX_EX
	GETN_CHECK
	GETN_LISTBOX GETN_MULTISEL_LISTBOX
	GETN_BUTTON_GROUP
	GETN_BUTTON
	GETN_COMBO(for numeric) GETN_LIST(for string, return the index of selection) GETN_STRLIST(for string, return the selected text) GETN_STR_GROUP(multiple selections)
	GETN_COMBO_BUTTON
	GETN_RANGE
	GETN_STR(for string)

	GETN_NUM(for numeric) GETN_MULTILINE_TEXT(multiple lines text)
	GETN_SPINNOR_DOUBLE
	GETN_SLIDER GETN_SLIDEREDIT(editable)
	GETN_COLOR
	GETN_SYMBOL
	GETN_SEPARATOR_LINE
	GETN_DATE
	GETN_TIME
	GETN_PASSWORD
	GETN_XYRANGE GETN_XYRANGE_COMPLEX GETN_XYZRANGE GETN_INTERACTIVE

18.2.3.3 Event Handler

In the above dialog box, you can add an event function like below function *node_event* to dynamicly show/hide a control, or reconstruct a combo list and so on.

Change the line in the above example function from

```
GetNBox(trRoot);
```

to

```
GetNBox(trRoot, node_event);
```

Add an event fucntion as below:

```
int node_event(TreeNode& trRoot, int nRow, int nEvent, DWORD& dwEnables,
        LPCSTR lpcszNodeName, WndContainer& getNContainer, string& strAux,
        string& strErrMsg)
{
    if( 0 == lstrcmp(lpcszNodeName, "type") || GETNE_ON_VALUE_CHANGE == nEvent
    || GETNE_ON_INIT == nEvent )
    {
        trRoot.Constant.Show = (0 == trRoot.type.nVal); // show Constant
        trRoot.Reference.Show = (1 == trRoot.type.nVal); // show reference
    }
    return 0;
}
```

18.2.3.4 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>
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 bShowApply = true;

    if(GetNBox(tr, NULL, "Example", NULL, GetWindow(), bShowApply,

                _apply_event))

    {

        out_str("Click OK");

    }

}


// 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;

}
```

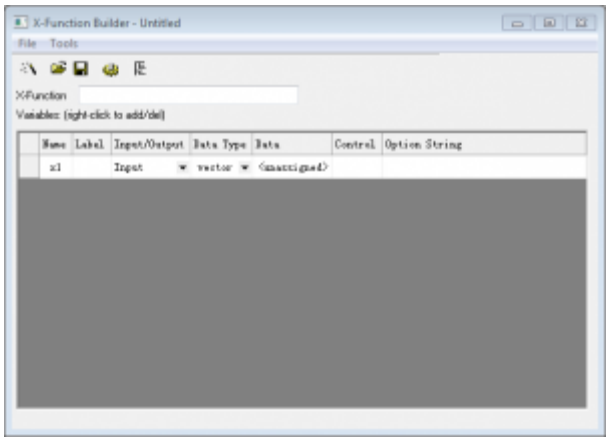


By default, the Apply button in a GetN dialog will become inactive after first click and won't be active again unless a change is made to the GetN dialog. If you want to keep the Apply button always active, set system variable @EAB = 0.

18.2.4 X-Function

18.2.4.1.1 Dialog generated by X-Function

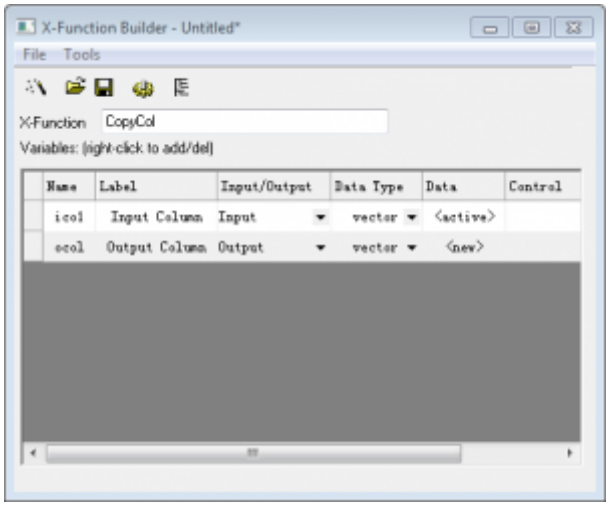
This example shows how to automate the dialog generating procedure by creating X-Function with X-Function builder. You can open X-Function Builder by choosing X-Function Builder on the Tools menu or by pressing F10:



18.2.4.1.2 Creating X-Function

The steps below will walk you through the creation of an X-Function that will perform the task of copying data from one column to another column:

With the X-Function builder opened, name the X-Function "CopyCol" and add a second variable by right-clicking on the Variables list and choosing "Add variable". Change the variable names, labels, and other values to match the settings in the dialog below.



2. After you make the necessary changes to the variables, save the X-Function by clicking the Save OXF file button . When the Save As dialog appears click the Save button.

3. Now we need to write the Origin C code that will do the work for our X-Function. Click on the Code Builder button . This will open the X-Function in Code Builder where we can write our Origin C code. In the main function, add the following Origin C code:

```
ocol = icol;
```

4. Click the Compile button and then click the Return to Dialog button to go back to X-Function Builder. Click the Save OXF file button.

18.2.4.1.3 Launch the X-function dialog

We can test our X-Function now:

Create a new a worksheet with two columns. Fill column A with row numbers and select the whole column by clicking on its heading.

In the Script or Command Window enter "CopyCol -d", without the quotes, and press Enter.

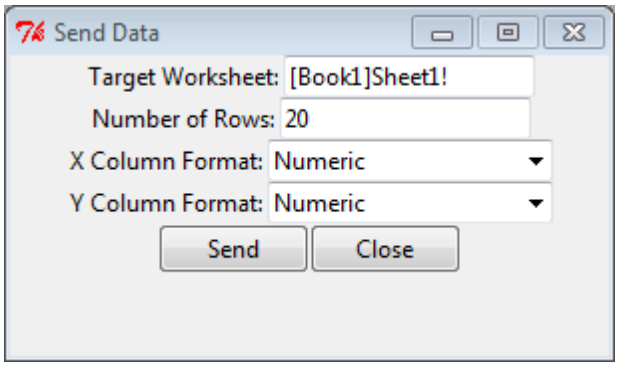
When the dialog appears, keep the default values and click the OK button.

After the X-Function executes, the worksheet will contain a third column which will contain a copy of the data from column A.

18.2.5 Python Dialog

18.2.5.1 Dialog created by Python

This example (\samples\Python\Run SendData toWks py.opj) demonstrates using the **tkinter** module in Python to generate an interactive graphic interface, so that users can specify the data type, the target worksheet and row # before importing data. To view the Python code, you can open the **Code Builder** (**View:Code Builder** or **ALT+4**) and browse to the **Project** folder in **Workspace** panel to access the attached Python file.



18.2.6 Dialog Builder

18.2.6.1 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.

Note: Beginning with Origin 2017, Origin C dialogs can be built using HTML and JavaScript. For most users, this will be a better approach than building tools using the older Origin Developer Kit. Building dialogs with the Developer Kit requires the user to create a resource DLL using Visual Studio. This has proven to be a barrier for many, both for development and for distribution. The HTML/JS approach eliminates such complications.

Dialog Builder Samples

Dialog Builder sample files, including resource DLLs, can be found in this zip file, under the *\Dialog Builder* subfolder.

This section covers the following topics:

Simple Hello World Dialog

Wizard Dialog

Graph Preview Dialog

Splitter Dialog

18.2.6.2 Simple Hello World Dialog

18.2.6.2.1 Create Resource DLL in VC

18.2.6.2.1.1 Create by Origin Dialog AppWizard

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.

Choose a simple dialog, and click Next.

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.

Click menu Build->Set Active Configuration to choose Debug or Release.

Choose menu Build->Builder ODialog.dll to create DLL.

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.

Open the ODialog.cpp file in Origin C Code Builder, compile, and run the DoMyDialog function to open the dialog.

18.2.6.2.1.2 Create by Win32 Dynamic-Link Library

This section describes how to create a Resource-only DLL in Visual C++ 6.0.

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.

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.

Select Insert->Resource to insert resources into the project. For a Dialog and controls on it, set dialog ID to IDD_OC_DIALOG.

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.

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.

Build the whole project with Debug or Release configuration. The resulting DLL file is generated under the Debug or Release subfolder.

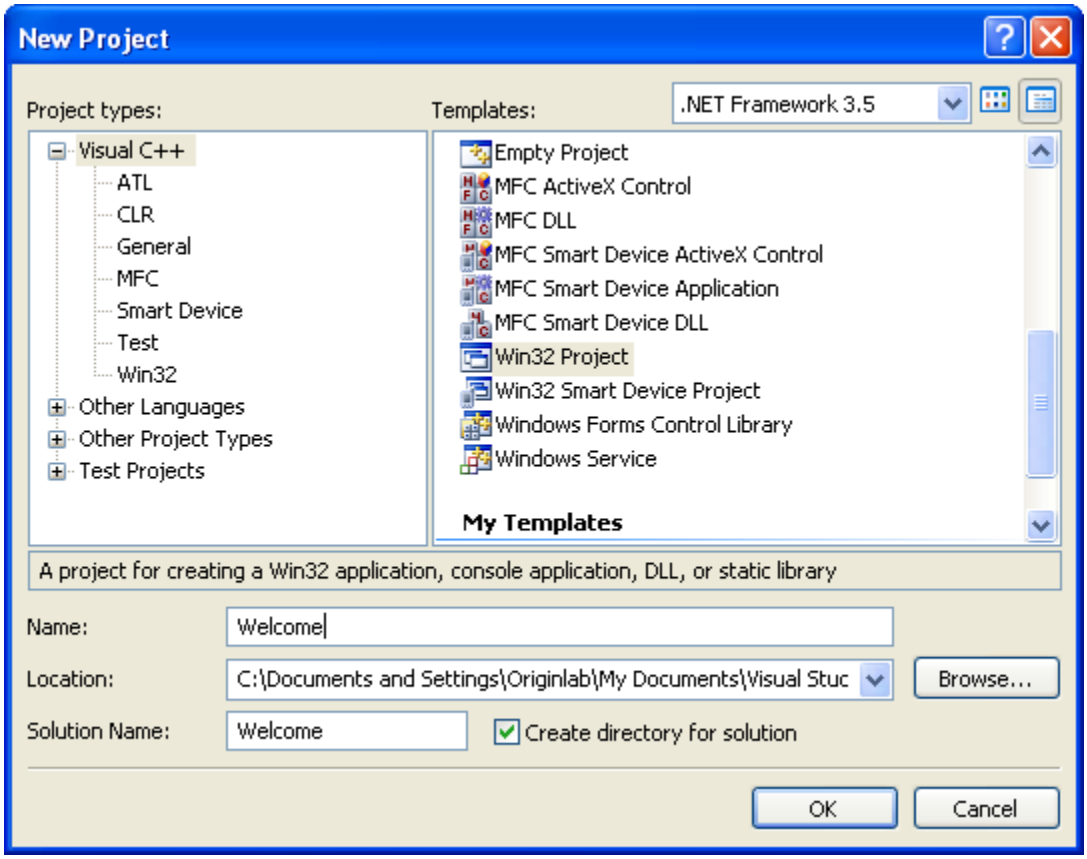
18.2.6.2.1.3 Create Resource-only DLL in Visual Studio 2008

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.

Start Microsoft Visual Studio 2008.

Select File->New->Project to create a new project.

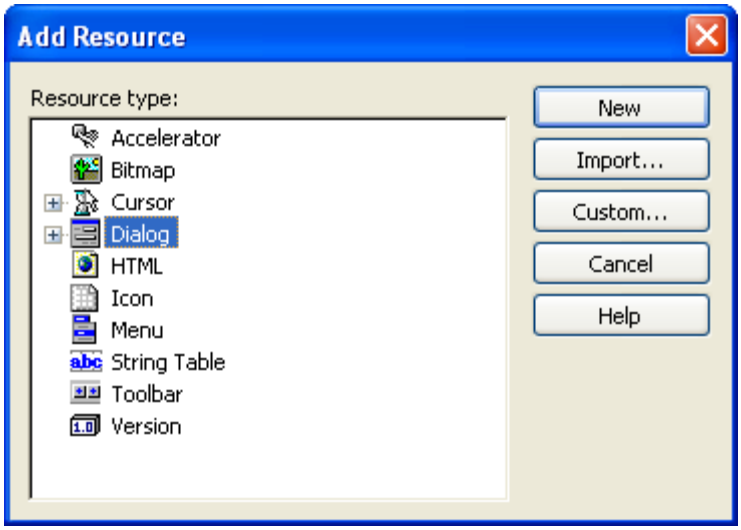
In the New Project dialog, choose Visual C++ as the programming language and Win32 Project as the template, type in the project name as "Welcome", select its location, like in the following picture, and click OK.



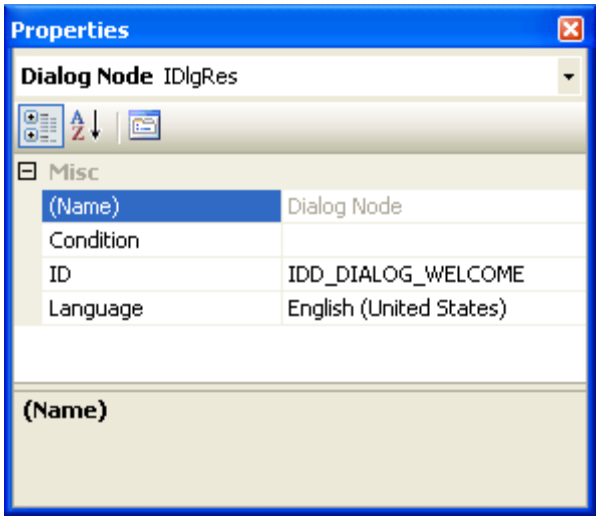
In the Win32 Application Wizard dialog, set Application type as DLL and click Finish.



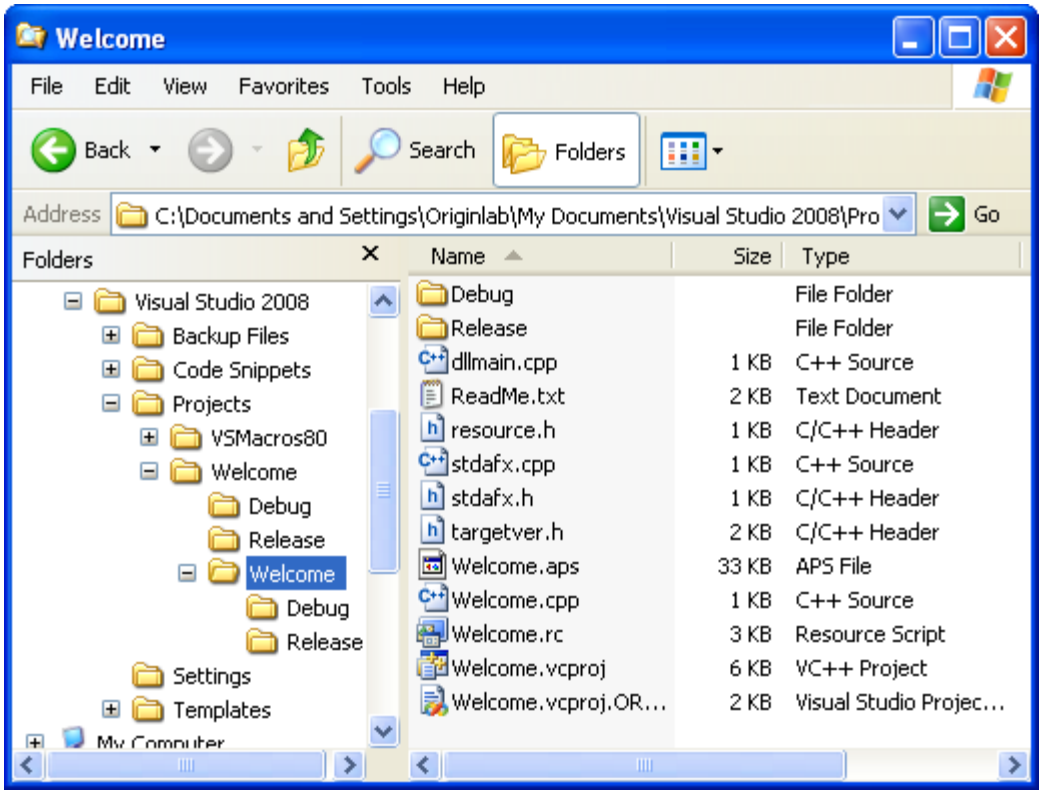
Switch to the Resource View of the project, right click on the project name to add resources, choose a resource type and click New.



Remember to set the Language property of the resource according to the environment in which your software will be installed; say English(United States) if your software is in English.



Add more controls as required, 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:



18.2.6.2.2 Use Resource DLL in Origin C

This section describes how to use the Resource-only DLL created in the section 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 resource.h file.

Start Origin and open Code Builder.

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();

}
```

18.2.6.3 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 this zip file, under *\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;
};
```

18.2.6.4 Graph Preview Dialog

This section shows how to create custom dialog with a graph preview.

18.2.6.4.1 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*.

18.2.6.4.2 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.

18.2.6.4.3 Including Needed Headers

```
//These headers contain declarations of dialog and controls
#include <..\Originlab\DialogEx.h>
#include <..\Originlab\GraphPageControl.h>
```

18.2.6.4.4 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;
};
```

18.2.6.4.5 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);
    }

protected:
EVENTS_BEGIN
        ON_INIT(OnInitDialog)
        ON_BN_CLICKED(IDC_LOAD, OnDraw)
EVENTS_END

    //message handler of dialog events
    BOOL    OnInitDialog();
    BOOL    OnDraw(Control ctrl);

private:
```

```
    //member stands for the preview control
    MyPreviewCtrl       m_Preview;
};


BOOL MyGraphPreviewDlg::OnInitDialog()
{
        m_Preview.Init(IDC_FB_BOX, *this);
        Button btn = GetItem(IDC_LOAD);
        if( btn )
                btn.Text = "Draw";
        return true;
}


BOOL MyGraphPreviewDlg::OnDraw(Control ctrl)
{
        vector vecX, vecY;
        vecX.Data(1.0, 10.0, 0.5);
        vecY.SetSize(vecX.GetSize());
        for(int ii = 0; ii < vecX.GetSize(); ++ii)
                vecY[ii] = rnd();


        m_Preview.Update(vecX, vecY);
        return true;
}
```

18.2.6.4.6 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.

18.2.6.5 Splitter Dialog

This example shows how to create a splitter dialog, which provides a better display of tree view or grid view.



18.2.6.5.1 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.

18.2.6.5.2 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.

18.2.6.5.3 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>
```

18.2.6.5.4 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;

}
```

18.2.6.5.5 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;
}
```

18.2.6.5.6 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());

}
```

18.2.7 Origin C HTML Dialog with JavaScript Support

18.2.7.1 Origin C HTML Dialog with JavaScript Support

HTML-Dialog

Origin C now supports using HTML to build dialog boxes in Origin without using a resource DLL. This not only removes the need to use Visual Studio to build a resource DLL but to allow making use of the vast array of utilities available in the public domain for web page constructions. In order to access and control the elements in the HTML dialog, JavaScript integration has been added in Origin 2017, which provides methods for Origin C calling JavaScript and JavaScript calling Origin C. In addition, an Origin graph control can overlap the HTML control so that both of them can be placed arbitrarily and well organized in the dialog.

This guide contains a tutorial that shows how to create an HTML page and then how to use Origin C to display the HTML page as a control in the dialog. Additional sections describe in more detail how to create an HTML dialog with a graph control and how does Origin C calling JavaScript works and vice versa.

Examples

Sample files of this tutorial, including the *.cpp file, the *.html file, the related *.css files and *.js file, are available for download

If you'd like to learn more about HTML dialog, please see this page for a nice collection of often useful examples.

This section covers the following topics:

Simple Hello World Dialog

Create a Calculator

HTML Dialog with a Graph

Floating Dialog to download file in another thread

HTML Dialog to use Chrome

Chrome HTML Dialog communicates to JavaScript



In Origin, the newHTML X-Function is provided to create a simple HTML dialog easily and quickly. The HTML file and CPP file created by this X-Function are like the templates, you can redesign your HTML dialog based on these two files. Once you go through all tutorials and have the basics about HTML dialog, you can use the newHTML X-Function to construct starter HTML dialog. In Code Builder interface, you can easily select menu **Tools:New HTML Dialog...** to launch this X-Function.

18.2.7.2 Simple Hello World Dialog

first-HTML-dialog

18.2.7.2.1 Summary

This tutorial will show how to:

Create an HTML page.

Display the HTML page in the dialog box by OriginC.

Build an event handler to respond to events of dialog.

Minimum Origin Version Required: Origin 2017

18.2.7.2.2 Sample Files

index.html: This is the HTML code for the html page.

HelloWorldDlg.cpp: This is the OriginC code to call out the HTML page and show it in the dialog.

HelloWorldDlg1.cpp: This is the OriginC code that includes more advanced features.

> **Note:** You can download the sample files here.

18.2.7.2.3 Creating an HTML Page

When you create an HTML dialog, the first step is creating an HTML page.

Start Origin and open **Code Builder** by clicking button .

Create a new HTML file named index.html and save it to a new folder named Hello World.

Inside the< head > and < body > elements, add the other HTML elements such as < title >, < h1 >, < p > to construct an HTML page.

```
<!DOCTYPE html>
<html>
    <meta charset="utf-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=Edge"/>
    <head>
       <title>
          A Small Hello
       </title>
    </head>
    <body>
        <h1>Hello World</h1>
        <p>This is very minimal "hello world" HTML document.</p>
    </body>
</html>
```

Note: When editing is complete, you can check the content of this page by opening it with any web browser.

18.2.7.2.4 Creating an HTML dialog

In this section, you will make the generated HTML page display in the dialog.

In **Code Builder**, create a new cpp file named HelloWorldDlg.cpp under the path of the HTML file.

Include three needed header files at first. Note that HTMLDlg.h is new in Origin 2017 which connects Origin and the HTML dialog together.

```
#include <Origin.h>
#include <..\OriginLab\DialogEx.h>
#include <..\OriginLab\HTMLDlg.h>
```

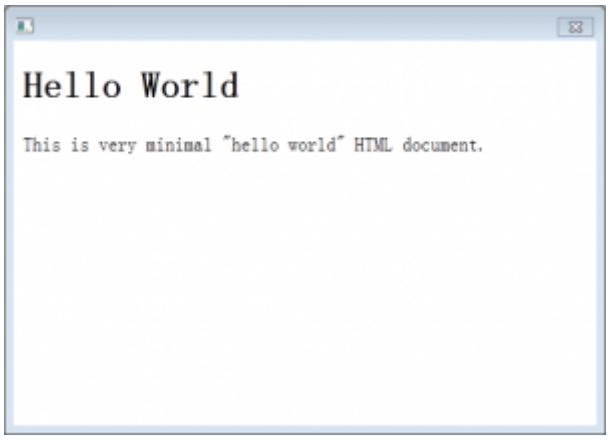
Derive a user defined HTML dialog class from the class `HTMLDlg`, and point to the HTML page inside the method `GetInitURL()`.

```
class HelloWorldDlg: public HTMLDlg
{
    public:
        string GetInitURL() //get the path of html file
        {
            string strFile = __FILE__; //the path of the current file
            return GetFilePath(strFile) + "index.html";
        }
};
```

To open the HTML dialog, save all the code and build it, then execute the following function.

```
void hello()
{
    HelloWorldDlg dlg;
    dlg.DoModalEx(GetWindow());
}
```

An HTML dialog with "Hello World" will pop up:



18.2.7.2.5 Advanced Feature

If you want to learn more, try to improve the program by modifying the dialog and adding event handlers.

> **Note:** The complete code of the following features is available in HelloWorldDlg1.cpp.

18.2.7.2.5.1 Preventing dialog resizing

To do this, use `ModifyStyle` to disable WS_THICKFRAME in the method `OnInitDialog` inside your dialog class:

```
BOOL OnInitDialog()
{
    ModifyStyle(WS_THICKFRAME, 0);//prevent resizing
    return HTMLDlg::OnInitDialog();
}
```

18.2.7.2.5.2 Setting dialog size

You need to call a parent class method `GetDlgInitSize` from you dialog class, and specify the width and height of the dialog in it:

```
BOOL GetDlgInitSize(int& width, int& height)//get dialog size
{
```

```
    width = 500;

    height = 200;

    return true;

}
```

18.2.7.2.5.3 Adding event handler

Similar to Dialog Builder, a message map is used to specify which events will be handled and which function will be called to handle them.
As an example, an event handler methods are added inside the user defined class HelloWorldDlg to pop up a message when the dialog is closed.

Map the dialog message by the following message map inside the class `HelloWorldDlg`.

```
//Message Map

public:

    EVENTS_BEGIN_DERIV(HTMLDlg)

        ON_DESTROY(OnDestroy)

    EVENTS_END_DERIV
```

Add an event handler method inside the class `HelloWorldDlg` to pop up a message box when the dialog is closed.

```
BOOL OnDestroy()

{

    MessageBox(GetSafeHwnd(), _L("Thank you and have a good day!")
,_L("Message"));

    return true;

}
```

18.2.7.3 Create a Calculator

second-HTML-dialog

18.2.7.3.1 Summary

In this tutorial, you build a calculator in which you can perform arithmetic operations (addition, subtraction, multiplication, and division).
You can learn how to:

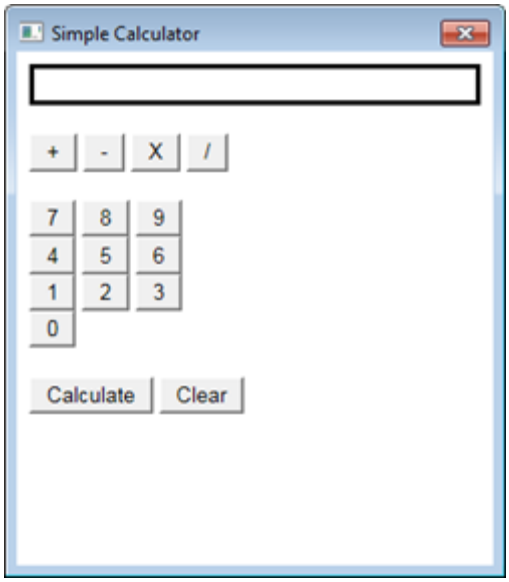
Trigger events to occur by using the controls inside HTML page.

Build event handler functions to respond to events by JavaScript.

Call OriginC function in JavaScript.

Perform basic arithmetic operations in OriginC.

When you finish, your calculator will look like the following picture:



Minimum Origin Version Required: Origin 2017

18.2.7.3.2 Sample Files

SimpleCalc.html: This is the HTML code for the html page.

Calculator.cpp: This is the OriginC code to call out the HTML dialog and perform arithmetic operations.

> **Note:** You can download the sample file here.

18.2.7.3.3 Creating an HTML Page for Calculator

As the first step in developing this calculator, you create an HTML page and add equation display box, number input buttons, operator buttons, Calculate button and Clear button to this page.

Start Origin and open **Code Builder**, create a new HTML file named SimpleCalc.html and save it to a new folder named Simple Calculator.

Within the SimpleCalc.html, construct a simple user interface for calculator by the following code:

```
<!DOCTYPE html>
<html>

    <head>
        <meta charset="utf-8" />
        <meta http-equiv="X-UA-Compatible" content="IE=Edge"/>
    </head>

    <body>
        <!--equation display textbox-->
        <div id="Equation" style="border:solid;height:20px;"></div>
        </br>

        <!--four arithmetic operations buttons-->
        <div id="Ops">
                <input type="button" id="btnPlus" value="+"/>
                <input type="button" id="btnMinus" value="-"/>
                <input type="button" id="btnMultiply" value="X"/>
                <input type="button" id="btnDiv" value="/"/>
        </div>
        </br>
```

```
        <!--number input buttons from 0 to 9-->
        <div id="Num789">
                <input type="button" id="btn7" value="7"/>
                <input type="button" id="btn8" value="8"/>
                <input type="button" id="btn9" value="9"/>
        </div>
        <div id="Num456">
                <input type="button" id="btn4" value="4"/>
                <input type="button" id="btn5" value="5"/>
                <input type="button" id="btn6" value="6"/>
        </div>
        <div id="Num123">
                <input type="button" id="btn1" value="1"/>
                <input type="button" id="btn2" value="2"/>
                <input type="button" id="btn3" value="3"/>
        </div>
        <div id="Num0">
                <input type="button" id="btn0" value="0"/>
        </div>
        </br>

        <!--calculation button and clear button -->
        <div>
                <input type="button" id="btnCalculate" value="Calculate"/>
                <input type="button" id="btnClear" value="Clear"/>
        </div>
```

```
    </body>


</html>
```

Note: When editing is complete, you can check this page by opening it with any web browser.

18.2.7.3.4 Adding Onclick Event Handlers for Buttons

The calculator works as follows:

When a user clicks number input buttons, the program shows the input number in the textbox.

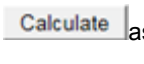
Then the user clicks an operator button, the program appends the operator to the number inputted before.

Next the user click to input another number that is also appended in textbox.

At last, the user click Calculate button to evaluate the equation and result in the textbox.

The text in textbox can be erase by clicking Clear button at any time.

In order to get your program to work that way, you need to add an `onclick` event handler to capture a click event from the users' mouse on the button in the HTML page. This action usually results in a call to a JavaScript function which helps to perform the calculation.

Taking the Calculate button 

as an example:

Inside the element < script >, add the function `Invoke_Cal()`. Note that this function will be used to respond to the click button event by `onclick` event handler mentioned in Step 2.

```
    <script>

        function Invoke_Cal() {}

    </script>
```

Add an onclick handler event inside the calculation button tag, so that the function `Invoke_Cal()` will be invoked if the user clicks this button.

```
<input type="button" id="btnCalculate" value="Calculate"
onclick="Invoke_Cal()"/>
```

Similar to the previous steps, you can add the `onclick` event handler inside every button to capture click event and add its corresponding JavaScript functions inside <script>.

18.2.7.3.5 Calling OriginC Function in JavaScript

In this example, arithmetic operations are performed in OriginC in order to show you how to call OriginC in JavaScript.

In JavaScript, you will use the function `window.external.ExtCall` to call into OriginC:

```
window.external.ExtCall("OriginC Function Name", Parameter1, Parameter2…)
```

> Note:
>
> Currently, no more than 5 parameters can be passed into ExtCall, including the OriginC function name.
>
> The parameters that can be passed as input arguments of OriginC should only be primitive types, including bool, integer, double and string.

Now, you can modify the functions inside <script> as follows after you figure out how to call OriginC in JavaScript, and the complete code is available in the HTML sample file.

```
<script>
    var PlusOp = "+";
    var MinusOp = "-";
    var MultiplyOp = "*";
    var DivOp = "/";

    function Invoke_AddOp(strOp) {
        var OriginStr = document.getElementById('Equation').innerHTML;
        //Calling OriginC function AddOp to show the operator symbol in textbox
        var NewStr = window.external.ExtCall("AddOp", OriginStr, strOp);
        document.getElementById('Equation').innerHTML = NewStr;
```

```
    }


    function Invoke_AddNum(NewNum) {

        var OriginStr = document.getElementById('Equation').innerHTML;

        //Calling OriginC function AddNum to show the number input in textbox

        var NewStr = window.external.ExtCall("AddNum", OriginStr, NewNum);

        document.getElementById('Equation').innerHTML = NewStr;

    }


    function Invoke_Cal() {

        var Str = document.getElementById('Equation').innerHTML;

        //Calling OriginC function Calculate to get the result of equation

        var Result = window.external.ExtCall("Calculate", Str);

        document.getElementById('Equation').innerHTML = Result;

    }


    function Invoke_Clear() {

        //Calling OriginC function Clear to erase the text in textbox

        document.getElementById('Equation').innerHTML =
window.external.ExtCall("Clear");

    }

</script>
```

18.2.7.3.6 Creating an HTML Dialog

In this section, you edit and execute OriginC code to launch the calculator dialogue in Origin.

In **Code Builder**, create a new cpp file named Calculator.cpp under the path of SimpleCalc.html

Include the needed headers.

```
#include <Origin.h>

#include <..\OriginLab\DialogEx.h>

#include <..\OriginLab\HTMLDlg.h>
```

Derive a user defined HTML dialog class from the class `HTMLDlg`

```
class OriginCalculatorDlg: public HTMLDlg
{
    public:
        string GetInitURL()
            {
                return GetFilePath(__FILE__) + "SimpleCalc.html";
            }
        string GetDialogTitle()
            {
                return "Simple Calculator";
            }
};
```

Add a main function `calc` which launches the calculator dialog.

```
void calc()
{
    OriginCalculatorDlg dlg;
    dlg.DoModalEx(GetWindow());
}
```

Save all the code and build it, then execute the `calc()` function to open the dialog.

18.2.7.3.7 Adding JavaScript Callable Prototypes in OriginC

In this section, you add methods in your dialog class `OriginCalculatorDlg`, which are called in JavaScript to perform the basic arithmetic operations.

Add DECLARE_DISPATCH_MAP in your dialog class, which help JavaScript to locate the functions

```
public:
    DECLARE_DISPATCH_MAP
```

Add the methods to perform arithmetic operations, show and clear the equation in textbox.

```
//when clicking any operator
string OriginCalculatorDlg::AddOp(string str, string strOp)
{
    return str + " " + strOp + " ";
}
//when clicking any digits
string OriginCalculatorDlg::AddNum(string str, int NewNum)
{
    return str + ftoa(NewNum);
}

//use LabTalk expression to evaluate
double OriginCalculatorDlg::Calculate(string str)
{
    double dd;
    LT_evaluate(str ,&dd);
    return dd;
}
```

```
string OriginCalculatorDlg::Clear(void)
{
    return "";
}
```

Declare the methods in your dialog class.

```
public:
    string AddOp(string str, string strOp);
    string AddNum(string str, int NewNum);
    double Calculate(string str);
    string Clear();
```

Map the methods of your OriginC dialog class to the HTML dialog

```
BEGIN_DISPATCH_MAP(OriginCalculatorDlg, HTMLDlg)
    DISP_FUNCTION(OriginCalculatorDlg, AddOp, VTS_STR, VTS_STR VTS_STR)
    DISP_FUNCTION(OriginCalculatorDlg, AddNum, VTS_STR, VTS_STR VTS_I4)
    DISP_FUNCTION(OriginCalculatorDlg, Calculate, VTS_R8, VTS_STR)
    DISP_FUNCTION(OriginCalculatorDlg, Clear, VTS_STR, VTS_VOID)
END_DISPATCH_MAP
```

Note: The syntax of DISP_FUNCTION should be:

```
DISP_FUNCTION(User-defined Dialog Class, Function Name, Type of
Ouput, Type of Input Type of Input…)
```

in which VTS_BOOL = bool, VTS_I4 = integer, VTS_R8 = double, VTS_STR = string and VTS_VOID = void.

Save the code and rebuilt it, then excute the calc() function again, you can do simple calculation on this calculator now.

18.2.7.4 HTML Dialog with a Graph

third-HTML-dialog

18.2.7.4.1 Summary

In this tutorial, you build a dailog works as follows:

There are three panels in the dialog, including a textbox in the left panel, a graph in the central panel and a blank right panel.

When a user moves the vertical line on the graph manually, the program shows the x-value in the textbox.

Conversely, when the user enters an x-value in the textbox and hit the ENTER key, the program moves the vertical line to the specified position.

You can learn how to:

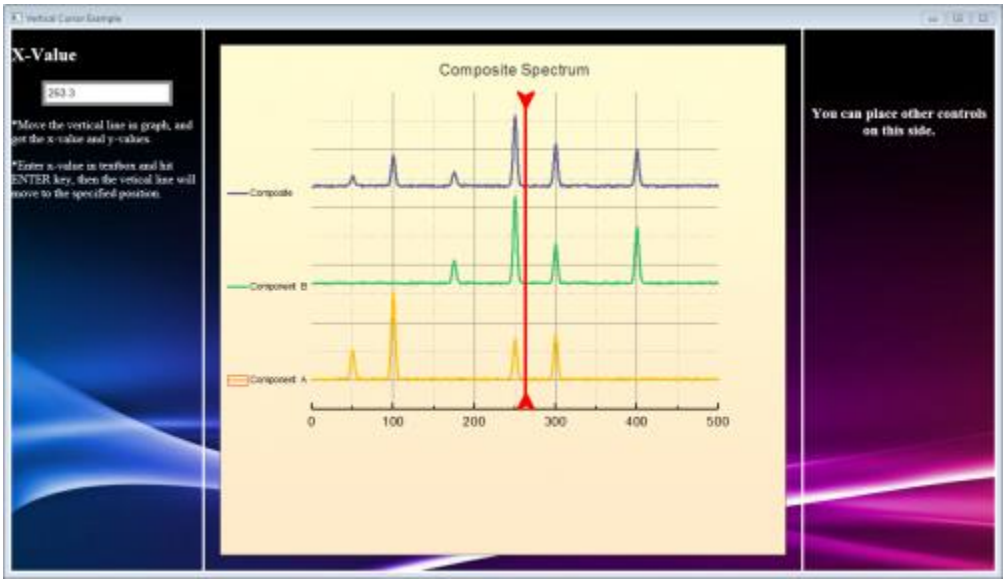
How to contain a graph in HTML dialog.

How to resize the dialog as you pleased.

How to call Origin C in JavaScript. (You can also learn this from the tutorial Create a Calculator.)

How to call JavaScript in Origin C.

When you finish this tutorial, your dialog will look like the following picture except the different number:



Minimum Origin Version Required: Origin 2017

18.2.7.4.2 Sample Files

Composite Spectrum.opj: This is the OPJ that contains a graph to be shown in the dialog

Index.html: This is the HTML code for the html page.

DlgWithGraph.cpp: This is the Origin C code to call out the HTML dialog and achieve the communication between Origin C and JavaScript.

Background_image1.png: This is the first picture for the background of dialog .

Background_image2.jpg: This is the second picture for the background of dialog.

> **Note:** You can download the sample file here

18.2.7.4.3 Preparation

New an Origin Project named Composite Spectrum.opj, and save it to the folder HTML Dialog with A Graph.

Select **Data: Import from File: Single ASCII** from Origin menu to import the data \Samples\Curve Fitting\Composite Spectrum.dat.

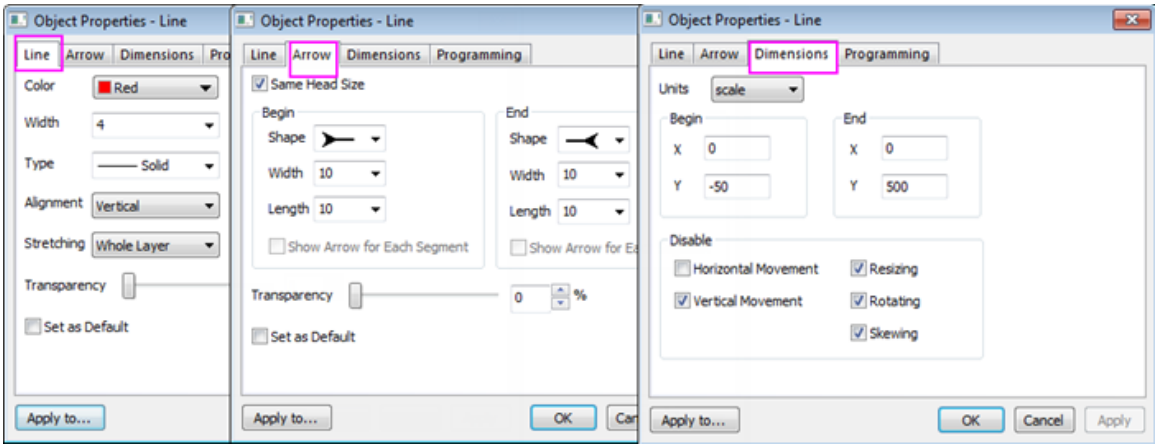
Highlight **Column B** to **Column D** and select **Plot: Multi-Y: Stacked Lines by Y** from menu to create a stacked line plot.

Double click on **Graph1** to open **Plot Details** dialog, and then improve the format of **Graph 1** as you pleased.

Click **Line Tool** button  to add a vertical line (press SHIFT key and draw) to the graph.

Right click on the vertical line and select **Properties…** in the context menu to bring up a dialog.

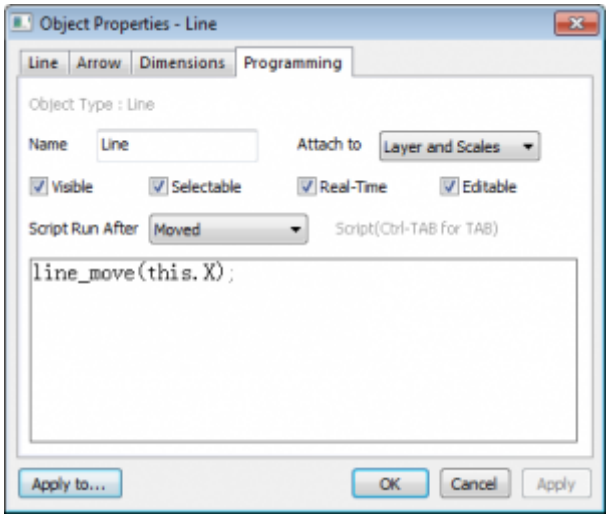
Go to **Line** tab and **Arrow** tab to modify the format of the vertical line, and then go to **Dimensions** tab to only allow it to move horizontally.



Go to **Programming** tab, name your line object **Line**.

Select **Moved** from the drop-down list of **Script Run After**, then type the following LabTalk script in the textbox.

```
line_move(this.X);
```



> **Note:** In this step, you set to trigger the function `line_move()` when a user moves the vertical line. This function `line_move()` will be created later in OriginC.

Open **Code Builder**, and then expand **Project** folder in **Workspace** window, double click ProjectEvents.OGS to open the file, and then add scripts inside the [AfterOpenDoc] section:

```
if(run.LoadOC("%X\DlgWithGraph.cpp", 16) == 0)
{
    @G=0;//graph background fill all so no gray band on either side
    HTMLandGraphDlgEx; //this is the function which launches the dialog
}
else
{
   type "Failed to load the dialog!";
   return 0;
```

```
}
```

> **Note:** The scripts help to launch the dialog after this Origin project is opened. If you want to learn more about ProjectEvents Script, please go to this page.

Save the ProjectEvents.OGS in **Code Builder** and save the project in Origin.

18.2.7.4.4 Creating an HTML Page for Dialog

When you create an HTML dialog, the first step is to design and create an HTML page for the dialog.

Prepare two pictures for the background in the folder HTML Dialog with A Graph.

Open **Code Builder**, create a new HTML file and save as index.html to the folder HTML Dialog with A Graph.

Copy the following code and paste it within the index.html:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
        <meta http-equiv="X-UA-Compatible" content="IE=Edge" />
            <style>
            body{
                background-image:url("background_image1.png")
            }
            input[type="text"] {
                font-size: 14px;
            }
            input[type="text"]:focus {
                outline: none;
```

```
}
.style-2 input[type="text"] {
    padding: 3px;
    border: solid 5px #c9c9c9;
    transition: border 0.3s;
}
.style-2 input[type="text"]:focus,
.style-2 input[type="text"].focus {
    border: solid 5px #969696;
}
#container {
    border: 3px solid #ffffff;
    position: absolute;
    left: 0;
    right: 0;
    bottom: 0;
    top: 0;
}
#leftcontainer {
    border-right: 3px solid #ffffff;
    width: 250px;
    height: 100%;
    float: left;
}
#rightcontainer {
    border-left: 3px solid #ffffff;
```

```
                width: 250px;

                height: 100%;

                float: right;

                text-align: center;

            }

        </style>

    </head>



    <body>

        <div id="container">

            <div id="leftcontainer">

                <h2><font color="ffffff">X-Value</font></h2>

                <ul class="input-list style-2 clearfix">

                    <input type="text" name="XValue"  id="X"
placeholder="X"  onkeydown = "">

                </ul>



                <p><font color="ffffff">*Move the vertical line in graph,
and get the x-value and y-values.</font></p>

                <p><font color="ffffff">*Enter x-value in textbox and hit
ENTER key, then the vetical line will move to the specified
position.</font></p>

            </div>



            <div id="rightcontainer">

                <h3>   </h3>

                <h3>   </h3>

                <h3><font color="ffffff">You can place other controls on
this side.</font></h3>
```

```
            </div>

        </div>

    </body>

</html>
```

Once you finish this step, you can see the following page by opening the index.html with a web browser:



18.2.7.4.5 Adding JavaScript Code for Dialog

In this step, you add JavaScript code.

Copy the following JavaScript code and paste it inside the element <script> of your html file.

```
<script>

    var vbg = 1;//counter to alternate html backgroud

    function lineMove(xVal)

    {

        var XValue = document.getElementById("X");

        XValue.value = xVal;

        //show html background change and graph not affected

        switch (vbg)

        {

            case 1:
```

```
                document.body.style.backgroundImage =
"URL('background_image2.jpg')";

                vbg = 2;

            break;

            case 2:

                document.body.style.backgroundImage =
"URL('background_image1.png')";

                vbg = 1;

            break;

        }

    }



    function enterXval()

    {

        var XValue = document.getElementById("X");

        window.external.ExtCall("OnEnterXvalToUpdateGraph", XValue.value);

    }



    function getGraphRect()

    {

        var leftDiv = document.getElementById("leftcontainer");

        var leftpos = leftDiv.getBoundingClientRect().right;

        var toppos = leftDiv.getBoundingClientRect().top;

        var bottompos = leftDiv.getBoundingClientRect().bottom;



        var rightDiv = document.getElementById("rightcontainer");

        var rightpos = rightDiv.getBoundingClientRect().left;
```

```
        return JSON.stringify({
            left: leftpos + 20,
            top: toppos + 20,
            right: rightpos - 20,
            bottom: bottompos - 20})
    }
</script>
```

Note: This code adds functionality to the HTML page you created above:

`lineMove()`:Update the x-value in textbox when a user moves the vertical line

`enterXval()`: Respond to the event of a user hitting the ENTER key in textbox

`getGraphRect()`: Return a rectangular box in which the graph is shown

In order to capture the hitting ENTER event, add an `onkeydown` event handler inside the textbox tag of HTML code, so that the function `enterXval()` will be invoked when the user hits the ENTER key.

```
<input type="text" name="XValue"  id="X" placeholder="X"  onkeydown =
"javascript:if(event.keyCode == 13) enterXval();">
```



In the function `lineMove()`, a switch statement is recommended to add. As the comment said, you can see the graph overlaps the HTML page and is not affected even you change the background.

In the function `enterXval()`, `window.external.ExtCall` is used to call Origin C. You can learn more about `window.external.ExtCall` from here.

18.2.7.4.6 Creating an HTML Dialog with Graph

Now you are ready to edit the Origin C code to create an HTML dialog with a graph.

In **Code Builder**, create a new cpp file named DlgWithGraph.cpp under the folder HTML Dialog with A Graph.

Include the needed headers.

```
#include <Origin.h>
#include <../OriginLab/DialogEx.h>
#include <../OriginLab/HTMLDlg.h>
#include <../OriginLab/GraphPageControl.h>
```

Define the GRAPH_CONTROL_ID.

```
#define GRAPH_CONTROL_ID 1
```

Derive a user defined HTML dialog class `HTMLandGraphDlg` from the class `HTMLDlg`, and point to the HTML page inside the method `GetInitURL()`.

```
class HTMLandGraphDlg: public HTMLDlg
{
    protected:
        string GetInitURL() //get the path of html file
        {
            string strFile = __FILE__; //the name of current file
            return GetFilePath(strFile) + "index.html";
        }
        string GetDialogTitle() {return "Vertical Cursor Example";} //set
the title of dialog
};
```

Add event handler methods inside the class `HTMLandGraphDlg` to respond to the event, which is triggered when a user opens, resizes or closes the dialog.

```
private:
    GraphControl     m_gcCntrl;
```

```
protected:

    BOOL OnInitDialog() //when the dialog is shown the first time, need to
init the dialog

    {

        LT_execute(";doc -m 0");//hide Origin main window

        HTMLDlg::OnInitDialog(); //Derive a dialog from HTMLDlg class

        ModifyStyle(0, WS_MAXIMIZEBOX);//the generic HTML dialog resource
did not have maximize button

        RECT rr;

        m_gcCntrl.CreateControl(GetSafeHwnd(), &rr, GRAPH_CONTROL_ID,
WS_CHILD|WS_VISIBLE|WS_BORDER);



        //set options to disable clicking on various components on a graph,
you can find more in OC_const.h

        DWORD   dwNoClicks = NOCLICK_AXES | NOCLICK_DATA_PLOT |
NOCLICK_LAYER | NOCLICK_TICKLABEL | NOCLICK_LAYERICON;

        GraphPage m_gp = Project.GraphPages("Graph1"); //this graph in the
OPJ can be customized manually

        //attach Graph1 in OPJ to GraphControl in dialog

        BOOL bb = m_gcCntrl.AttachPage(m_gp, dwNoClicks);

        return true;

    }



    BOOL OnDestroy()

    {

        //for the developer, this allow the OPJ to be modified,

        //and you can remove this part for the end users and just do
LT_execute(";doc -ss;exit;")

        bool bExitOrigin = MessageBox(GetSafeHwnd(), _L("Are you sure you
want to exit Origin?") ,_L("Vertical Cursor Example"), MB_YESNO) == IDYES;

        if (bExitOrigin)
```

```
            LT_execute(";doc -ss;exit;"); //exit Origin

        else

            LT_execute(";doc -m 1"); //show Origin main window

        return true;

    }



    //virtual

    // when the dialog is ready, need to init the size and position of
dialog

    BOOL GetDlgInitSize(int& width, int& height)

    {

        width = 1024;

        height = 450;

        return true;

    }

    // when you resize the dialog, need to reinit the size and position of
each control in dialog

    BOOL OnDlgResize(int nType, int cx, int cy)

    {

        if ( !IsInitReady() )

            return false;

        // MoveControlsHelper _temp(this); // you can uncomment this line,
if the dialog flickers when you resize it

        HTMLDlg::OnDlgResize(nType, cx, cy); //place html control in dialog



        if ( !IsHTMLDocumentCompleted() ) //check the state of HTML control

            return false;
```

```
        RECT rectGraph;

        //GetGraphRECT is a private function calling JavaScript to get a
rectangular box to involve the graph control.

        //You will get to how to call JavaScript in Origin C in the next
section.

        if ( !GetGraphRECT(rectGraph))

            return false;

        //overlap the GraphControl on the top of HTML control

        m_gcCntrl.SetWindowPos(HWND_TOP, rectGraph.left,
rectGraph.top,RECT_WIDTH(rectGraph),RECT_HEIGHT(rectGraph), 0);



        return true;

    }
```

> **Note:** The graph will be attached in a new-created graph control when the method `OnInitDialog()` is triggered.

Map the dialog message inside the class `HTMLandGraphDlg` to specify which events will be handled and which function will be called to handle.

```

EVENTS_BEGIN_DERIV(HTMLDlg)

    ON_INIT(OnInitDialog)

    ON_DESTROY(OnDestroy)

    ON_SIZE(OnDlgResize)

    ON_RESTORESIZE(OnRestoreSize)

EVENTS_END_DERIV
```

18.2.7.4.7 Calling JavaScript in Origin C

In this section, you need OriginC calling JavaScript to complete these two items:

The graph control should be moved to a right position along with the central panel when the user resizes the dialog, so you should call JavaScript to return a rectangular box in time.

The x-value in textbox should be updated when the user moves the vertical line, thus you should also call JavaScript to update the number.

In these section, you will use a member function `GetScript()` of the `DHtmlControl` class to get a script engine interface.

Add the function `GetGraphRect()` inside the class `HTMLandGraphDlg` to call the JavaScript function `getGraphRECT()` you wrote in the fifth section:

```
private:

    BOOL GetGraphRECT(RECT& gcCntrlRect) //this is the function to call
JavaScript and get the position of GraphControl

    {

        if (!m_dhtml)

            return false;

        Object jsscript = m_dhtml.GetScript();



        if(!jsscript) //check the validity of returned COM object is always
recommended

            return false;



        string str = jsscript.getGraphRECT();

            JSON.FromString(gcCntrlRect, str); //convert a string to a
structure

        return true;

    }
```

Add the function `OnMoveVlineToUpdateHtml()` inside your Origin C class to call the function `lineMove()` in JavaScript:

```
public:
```

```
    void OnMoveVlineToUpdateHtml(double dVal)

    {

        Object jsscript = m_dhtml.GetScript();

        if(!jsscript)

            return;

        string strXValue = ftoa(dVal, "*5*");//limit to 5 significant
digits and remove trailing zeros

        jsscript.lineMove(strXValue);

    }
```

18.2.7.4.8 Adding JavaScript Callable Prototypes in Origin C

In this section, you add a method `OnEnterXvalToUpdateGraph()` inside your dialog class `HTMLandGraphDlg`, which is called in JavaScript function `enterXval()` in order to move the vertical line when the user enter an x-value and hit the ENTER key.

Add the method and DECLARE_DISPATCH_MAP inside your dialog class.

```
public:

    DECLARE_DISPATCH_MAP


    void OnEnterXvalToUpdateGraph(string strXValue)

    {

        GraphPage gp = m_gcCntrl.GetPage();

        if(!gp)

            return;

        GraphLayer gl = gp.Layers(0);

        if(!gl)

            return;

        double xVal = atof(strXValue);
```

```
        if(NANUM == xVal)//user has not entered pure numeric
            return;
        GraphObject vline;
        vline = gl.GraphObjects("Line"); //access the vertical line in the
graph
        if(!vline)
            return;
        vline.X = xVal;
    }
```

Map the method of your OriginC dialog class to this HTML dialog.

```
BEGIN_DISPATCH_MAP(HTMLandGraphDlg, HTMLDlg)
    DISP_FUNCTION(HTMLandGraphDlg, OnEnterXvalToUpdateGraph, VTS_VOID,
VTS_STR)
END_DISPATCH_MAP
```

> **Note:** More details of DISP_FUNCTION can be found here.

18.2.7.4.9 Adding Labtalk Callable Prototypes in Origin C

You've almost created a nice dialog, but it needs an additional item to finish it.

You must remember that you have written a script to call a function `line_move()` in LabTalk when you set your vertical line properties in **Properties** dialog.

Now you should add this function in OriginC to be triggered when the vertical line is moved:

```
static HTMLandGraphDlg* s_pDlg = NULL;// we need this such that dialog class can be
used from LabTalk

//this is the OC function to be called from LabTalk
void line_move(double dVal) //when the line moves, get the X-value and pass the
value to JavaScript
```

```
{
   if ( s_pDlg ) //trigger only if move the line in this dialog
                s_pDlg->OnMoveVlineToUpdateHtml(dVal);
}
```

Note: In order for LabTalk to call the method within this dialog class, we need to declare a static `HTMLandGraphDlg` object.

18.2.7.4.10 Launching The Dialog

You are ready to launch the dialog.

Add a main function:

```
void HTMLandGraphDlgEx()
{
    HTMLandGraphDlg dlg;
    s_pDlg = &dlg;
    dlg.DoModalEx(GetWindow());
    s_pDlg = NULL;
}
```

Note: You should name the function the same with you wrote in ProjectEvents.OGS.

Save all the code and build it.

Now, you double click the OPJ directly to lauch the dialog.

18.3 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
    }
```

```
}
```

18.4 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);

        }

}
```

18.5 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

}
```

19 X-Functions

19.1 X-Functions

X-Function technology provides a framework for building Origin tools. In fact, most of Origin's revamped and greatly-expanded lineup of analysis and data processing tools have been implemented using this new technology.

19.2 Creating an X-Function

19.2.1 Creating an X-Function

Creating an X-Function is a two-step process: first, the X-Function and its parameters are defined in **X-Function Builder**; second, the X-Function's Origin C code is written and compiled in **Code Builder**.

This section covers the following topics:

Creating a Simple X-Function

X-Function Builder

Programming in Code Builder

Creating Graph Preview GetN Dialog

Creating Image GetN Dialog

Creating Custom Dialog

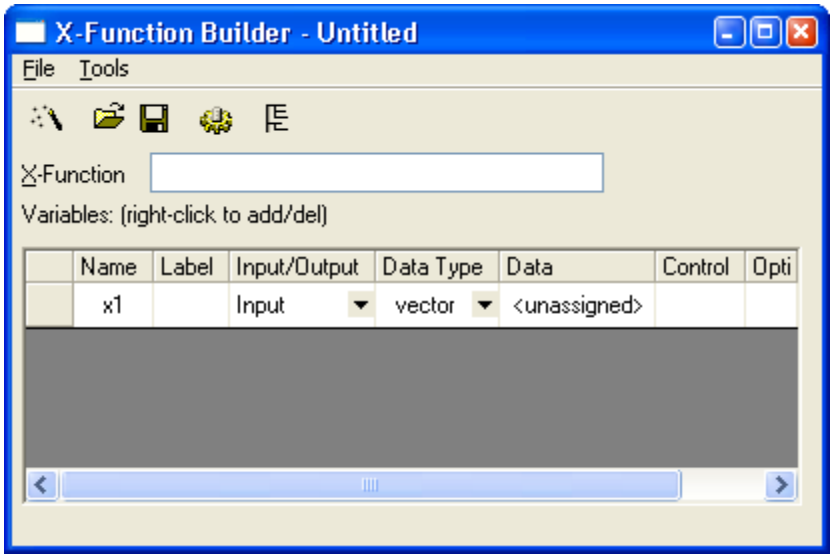
Creating Wizard Dialog

Creating a Gadget Tool with X-Function

19.2.2 Creating a Simple X-Function

X-Functions are created with X-Function Builder. You can open X-Function Builder by choosing X-Function Builder on the Tools menu or by pressing F10.

The X-Function Builder dialog is shown below.

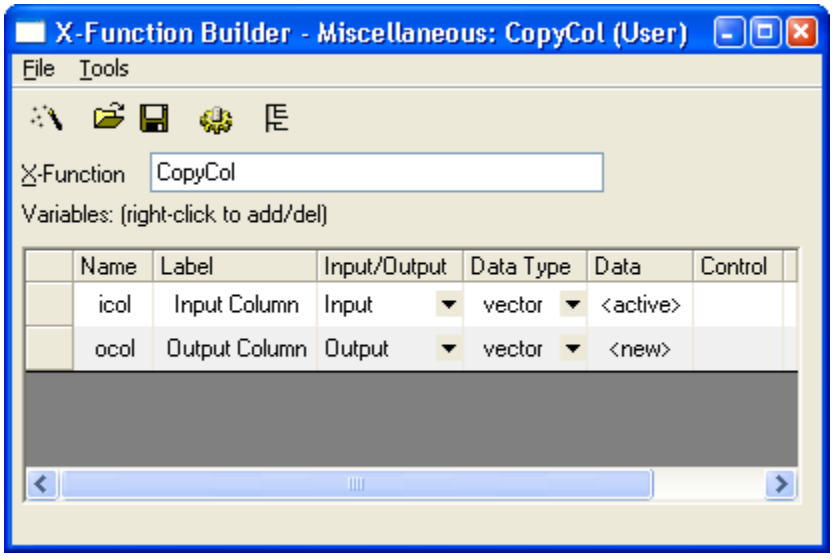


19.2.2.1 Creating X-Function

The steps below will walk you through the creation of an X-Function that will perform the task of copying data from one column to another column.

Start creating an X-Function by pressing F10 to open X-Function Builder.

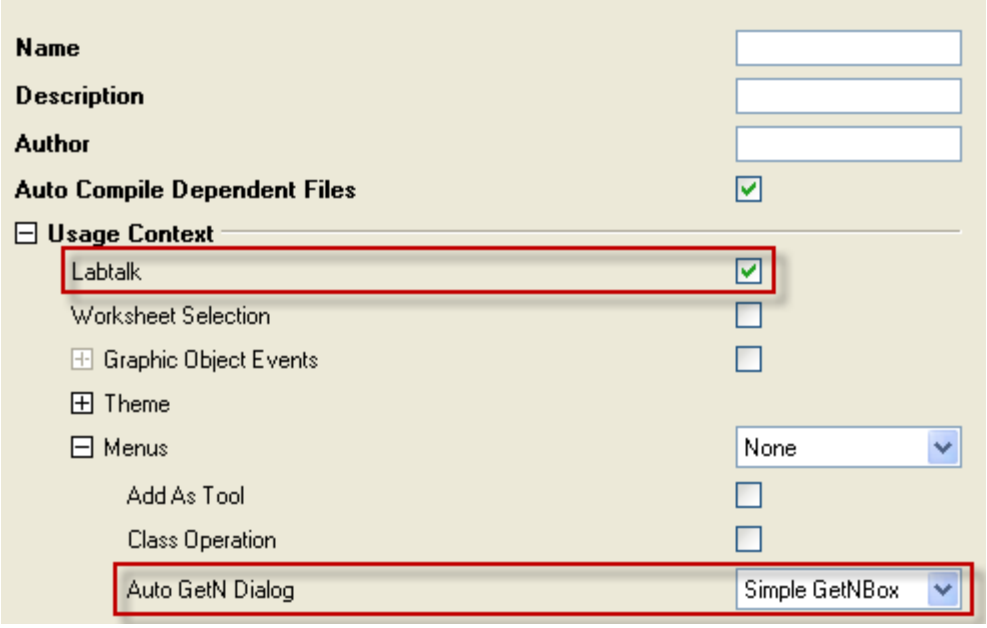
Name the X-Function "CopyCol" and add a second variable by right-clicking on the Variables list and choosing "Add variable". Change the variable names, labels, and other values to match the settings in the dialog below.



After you make the necessary changes to the variables, save the X-Function by clicking the Save OXF file button . When the Save As dialog appears click the Save button.

Click the X-Function Tree View button  to switch to tree view. Make sure the Labtalk check box is checked and the Auto GetN Dialog is set to Simple GetNBox, as below. Select the Labtalk check box to make it so the X-

Function can be used in the Command Window. The Auto GetN Dialog provides the choice to use an X-Function without dialog or with dialog, as there are multiple different types.



Now we need to write the Origin C code that will do the work for our X-Function. Click on the Code Builder button . This will open the X-Function in Code Builder where we can write our Origin C code. In the main function, add the following Origin C code:

```
ocol = icol;
```

Click the Compile button and then click the Return to Dialog button to go back to X-Function Builder. Click the Save OXF file button.

19.2.2.2 Using X-Functions

Now that we have successfully created our X-Function, we can test it.

Create a new a worksheet with two columns. Fill column A with row numbers and select the whole column by clicking on its heading.

In the Script or Command Window enter "CopyCol -d", without the quotes, and press Enter.

When the dialog appears, keep the default values and click the OK button.

After the X-Function executes, the worksheet will contain a third column which will contain a copy of the data from column A.

19.2.3 X-Function Builder

Open X-Function Builder by choosing X-Function Builder on the Tools menu or by pressing F10.

19.2.3.1 X-Function Name

Input the X-Function name in the X-Function edit box.



Do not give your X-Function the same name as an existing LabTalk macro, OGS file, LabTalk command, or LabTalk callable Origin C function.

If two or more of these have the same name, the priority order of calling is: LabTalk macros > LabTalk OGS > X-Function > LabTalk callable Origin C functions > LabTalk commands.

19.2.3.2 X-Function Variables

Name	Specify the name of the variable. When the X-Function is executed by script, you have to assign values to variables using their names. Therefore, we recommend that you use shorter names and begin them with different characters.
Label	Specify a label for the variable. Special characters and white space are allowed. Displays on the X-Function dialog as control label. If not specified, variable name will be used.
Input/Output	Specify whether the variable is used for input, output or both.
Data Type	Specify the variable data type. More details can be referred to below in the Variable Data Type and Default Value section.
Data	Specify the default value for the variable. More details can be referred to below in the Variable Data Type and Default Value section.
Control	Specify the control for the variable in the dialog box (combo box, slider, etc.). More details can be referred to below in the Control Types section.
Option String	Specify an option string for the variable. You can apply advanced settings with option strings, such as grouping several variables together, enabling recalculation, etc. More

	details can be referred to below in the Option String section.
Theme ID	Specify the Theme ID for the variable. If not assigned, the Theme ID of the variable will refer to the index of the variable. The Theme ID is used in the theme file for the X-Function dialog. More details can be referred to below in the Theme ID section.

19.2.3.3 Variable Data Type and Default Value

X-Functions provide dozens of data types, as below.

Basic Data Type

Types are double, string, int.

The default value can be 0 for numeric or kept empty for string type.

Origin Composite Data Type

Types are vector, vector<string>, vector<complex>, and matrix.

For example, a vector can attach to a worksheet column as input or output.

The default value can be <active>, <new>, or <unassigned>.

For more details, please refer to the Origin C help Composite Data Types book.

Origin Object Type

The types are Column, Range, GraphLayer and other Origin Object types.

The default value can be <active>, <new>, <input>, or <unassigned>.

Origin C provides the function *BOOL okxf_resolve_string_get_origin_object(LPCSTR lpcsz, OriginObject* pObj)* to get an Origin Object from a GetN tree string value.

For more details, please refer to the Origin C help Internal Origin Objects book.

TreeNode Type

Types are TreeNode, ReportData, and ReportTree. TreeNode is input, ReportData and ReportTree are output.

Origin Graphical Object Control

Types are ButtonInfo, LineInfo, TextInfo and so on.

To see the usage, open a built-in X-Function from OriginEXE\X-Functions\Data Manipulation\curve translate.OXF in X-Function Builder as an example.

Report Type

Types are ReportData, and ReportTree.

See X-Function ReportTree Example in the X-Function Examples chapter.

19.2.3.4 Control Types

After specifying the different control strings in the variables grid Control column in the X-Function Builder, we can easily create different types of user interface controls in the X-Function dialog.

Usage	Data Type	Control String
Numeric Edit Box	double	
Text Edit Box	string	
Check Box	int	0|1
Combo Box	int	string1|string2|string3 CaseA:string1|CaseB:string2|CaseC:string3

Editable Combo Box	string	Keep empty in the Control column in the Variables grid, but add the codes below to the *xfname_before_execute function* to set the **var** variable control as an editable combo box.
		`if( nGetNDialog > 0 )`
		`trGetN.var.SetAttribute(STR_COMBO_ATTRIB, "|aa|bb|cc");`

Radio Box	int	Radio:string1|string2|string3

Slider	int/double	slider:start|end|nstep
Color List	int	ColorList
Graph Browser	string	*...Graph* for one graph *...Graphs* for multiple graphs Origin supports interface settings for the Graph Browser in an OriginC function, using the control string GraphBrowserFilter. See X-Function Graph Browser Dialog Example for details.
Text Edit Box with File Browser Button	string	*...File* for single file *...Files* for multiple files
Text Edit Box with Path Browser Button	string	...Path
Text Edit Box with Save As Browser Button	string	...Save

19.2.3.5 Option String

We can set the special string in the Option String column to specify the action of a variable. For example, setting the recalculate default mode, setting the number of decimal digits for a numeric control, or setting the default name of an output object.

19.2.3.6 Theme ID

The Theme ID is used in the X-Function dialog theme file, and the default value is 0. If you keep the Theme ID of all the variables at 0(default), the Theme ID in the theme file will be converted according to the index of the variable, beginning from 1.

When you want to insert a variable into an existing X-Function (to append a variable as the last, there's no need do this), in order to remain compatible with the existing theme file, you need to:

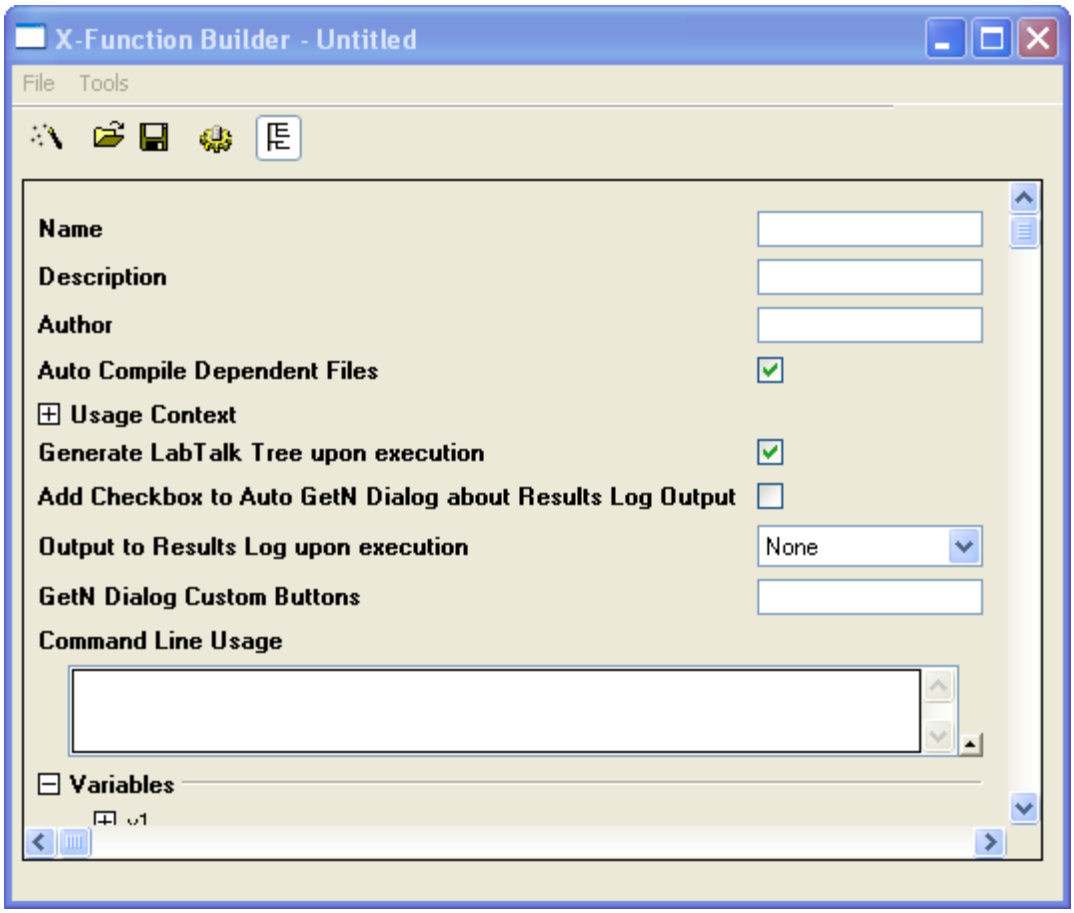
Before inserting the variable, change the Theme ID from 0 to the index of the variable.

Insert the variable.

Set the Theme ID for the inserted variable as the maximum to avoid repeats.

19.2.3.7 Tree View

Click the Tree View button  to switch to tree view. The settings in Tree View are mainly used to create X-Function dialogs, add X-Functions to the Origin menu, and add documentation. See the next chapter, Customizing an X-Function, for details.

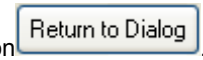


19.2.4 Programming in Code Builder

Click on the Code Builder button  to open the X-Function in Code Builder, where we can write our Origin C code.

19.2.4.1 Editor Window

There are two buttons near the top of the editor window: a Compile button 

and a Return to Dialog button 

. The Compile button is used to compile your X-Function's Origin C code. The Return to Dialog button will close the editor window and return you to the X-Function Builder dialog.

In the editor you will notice some of the Origin C code has a gray background. These parts of the code are read-only and cannot be changed. The X-Function framework generated these parts of the code and they are in a required syntax.

19.2.4.2 Include Header Files

You can include additional header files in your X-Function's Origin C code. While not required, it is recommended that you include your additional header files near the top of the file, after the following line:

```
//put additional include files here
```

This recommendation is to help keep the file organized and to be consistent with other X-Functions.

19.2.4.2.1 Search Paths

When you include files using the < and > brackets, then those files are included from the Origin C System folder located in the Origin installation folder. You can still provide a relative path to move above the System folder. These brackets behave the same when compiling .c and .cpp files in Origin C.

```
// Include fft_utils.h located in the "OriginC\System" folder.
#include <fft_utils.h>

// Include abffiles.h located in the "OriginC\OriginLab" folder
#include <..\OriginLab\abffiles.h> // relative path
```

When you include files using double quotes, then those files are included from the "OriginC\X-Functions" folder in your "User Files" folder. This behavior is different than when you compile a .c or .cpp file in Origin C. This different behavior is necessary because X-Functions are installed in a location maintained by Origin.

```
// Include myOtherHeader.h located in the "OriginC\X-Functions"
// folder of the "User Files" folder.
#include "myOtherHeader.h"
```

If you specify an absolute path between double quotes, then that file will be included regardless of where the file is located.

19.2.4.2.2 Auto Compile Dependent Files

In X-Function Builder's Tree View there is an option named Auto Compile Dependent Files. When this option is turned on it will cause the compiler to check for a .c or .cpp file sharing a name with an included .h file. If a .c or .cpp file is found then it will be loaded and compiled too.

```
// Include graph_utils.h located in the "OriginC\Originlab" folder
// If Auto Compile Dependent Files is turned on
// then graph_utils.c, located in the same folder, will be loaded and compiled
#include <..\Originlab\graph_utils.h>
```

19.2.4.3 Define Internal Functions

In order to provide more clarity and readability, to make your code easier for future understanding and maintenance, it is recommended that you break a long function into functionally independent and well-structured short functions, which is also proposed in some other programming languages. In an X-Function, you can add these short functions in as internal functions with the keyword *static* after the following line.

```
//put your own support static functions here
```

For example, in many X-Functions with lots of interaction from the GUI, many codes will be needed in *xfname*_event1( ), and then you can add a static function, similiar to Origin's interp1 X-Function.

```
static void _update_method_show(TreeNode& trGetN)
```

As you can see, it is also recommended that you use an underscore to separate the components in a function name.

19.2.4.4 Main Function

The main function is where you put all the code that performs the primary tasks of your X-Function.

Your X-Function's main function will have the same name you gave your X-Function in X-Function Builder. For example, if you named your X-Function **myXFunc** then the main function will also be named **myXFunc**.

The arguments passed to the main function are the variables listed in the X-Function Builder dialog. All the variables set as Input are passed in as constant references while the variables set as Output or Input/Output are passed in as writable references.

19.2.4.5 Event Handling Functions

Your X-Function's source code contains functions used for handling various events. The name of each event handler function is prefixed with the X-Function's name followed by an underscore character followed by the event name. For example; if your X-Function's name is **myXFunc** then the **before_execute** event handler function will be named **myXFunc_before_execute**.

The following is a list of events:

before_execute

event1

event1_ex

make_tree

By default all the event handler functions are empty. You only need to add code to an event handler function when you want to change the default behavior or do some additional processing during an event.

The following sections provide more details about each of the event handler functions including their purpose, parameters passed in, and return type.

19.2.4.5.1 before_execute

This function will be called before execution of the main function and will be called two or three times.

Prototype

```
void xfname_before_execute(TreeNode& trGetN, int nGetNDialog, int& nRet, int
dwCntrl)
```

Parameters

trGetN

A TreeNode containing the variables passed to the X-Function.

nGetNDialog

An integer value indicating when the function is being called.

A value greater than zero indicates the function is being called before showing its dialog. This happens when choosing **Open Dialog** from a menu, programmatically calling the X-Function with the **-d** option, or choosing **Change Parameters** for a manual recalculate.

Value	Before Opening Dialog for...
1	Simple GetNBox
2	GetNGraphBox with Preview
3	GetNImageBox

A value of -1 indicates the dialog was closed by the user clicking the OK button.

A value of zero indicates the main function is about to be called.

nRet

This option can control the execution states of the X-Function.

```
XFEVT_PROCEED         = 0 // Follow with normal execution
XFEVT_ABORT           = 1 // Abort execution
XFEVT_PROCEED_NO_DLG  = 2 // Execute silently without opening dialog.
```

dwCntrl

This option can be used to determine how this X-Function is accessed. Possible values are enumerated as:

```
LTXF_FROM_GUI_MENU              // Accessed from menu
LTXF_FROM_GUI_PROMPT            // Accessed from command window
LTXF_FROM_AUTO_UPDATE           // Accessed for Auto-update
```

```
LTXF_FROM_FUNCTION              // Accessed from Origin C function

LTXF_THEME_USED                 // Theme is used

LTXF_CHANGE_PARAM               // Accessed for Change Parameters
```

The following example shows how to check if the X-Function is accessed from the command window and a theme is also used (type ***xfname* -t *themename*** in the command window).

```
if(!(dwCntrl & LTXF_FROM_GUI_PROMPT) && (dwCntrl & LTXF_THEME_USED))

{

    // handle this case

}
```

19.2.4.5.2 make_tree

This function is called once for each TreeNode variable passed to the X-Function. The calls occur before the **before_execute** function is called.

Prototype

```
int xfname_make_tree(TreeNode& tr, LPCSTR lpcszVarName)
```

Return

Return zero to update the dialog or -1 to not update the dialog. The default behavior returns -1.

Parameters

tr

The TreeNode to be created, which should have been defined as an input TreeNode type variable of the X-Function.

lpcszVarName

The name of the TreeNode variable.

19.2.4.5.3 event1 and event1_ex

These are the primary event handling functions. They will be called by any event, such as dialog initialization, changes to any settings in the dialog, button clicking, and more. These functions are only used in X-Functions

with a simple GetN dialog. X-Functions with a graph preview or an image preview use a different set of event handler functions and are discussed in a later section.

Prototype

```
int xfname_event1(TreeNode& trGetN, int nRow, int nEventID, DWORD& dwEnables,
    LPCSTR lpcszNodeName, WndContainer& DynaCntrlContainer,
    string& strAux, string& strErrMsg)
```

Return

If returning true, the dialog will be updated.

Parameters

trGetN

This is the tree node of the variables defined in the X-Function.

nRow

The index of the row of the value-changed control node in TreeNode *trGetN*. The value will be -1 when the event is not 'control value changed'.

nEventID

Event type ID. For example, GETNE_ON_INIT, GETNE_ON_THEME.

dwEnables

This parameter is used to enable or disable the OK, Apply or other custom button. It is true by default, and will usually be set to false when the user's settings are invalid, so as to disable the OK or Apply button.

lpcszNodeName

If the event is generated by a control change in the dialog, it will be the variable name of this control. Otherwise, it will be an empty string.

DynaCntrlContainer

X-Function dialog window container.

strAux

This parameter is the auxiliary string, which is for advanced usage. Please refer to the header file <Origin Installation Directory>\OriginC\System\GetNBox.h for more details like the following definition.

```
#define DLG_NEED_CHANGE_GRID "GETN_CHANGE_GRID"

#define GETN_CHANGE_GRID  set_aux_value(strAux, DLG_NEED_CHANGE_GRID);
```

strErrMsg

This is the string message that will be shown in the bottom of the dialog. It is usually used to notify the user of any errors that occurred during the input of variable values.

19.2.4.5.4 GetNGraphPreview Event Handler Functions

The **event1** and **event1_ex** event handler functions are not used by X-Functions that have a graph preview dialog. Instead you use the **GetNGraphPreview** functions.

GetNGraphPreview_OnInitGrid

This event function is used to initialize the controls on the dialog.

GetNGraphPreview_OnInitGraph

This event function is used to initialize the preview graph.

GetNGraphPreview_OnUpdateGraph

This event function is used to update the preview graph on other events, for example, GUI changed.

GetNGraphPreview_OnChange

Any change will trigger this event function, and this function will decide whether to update the preview graph or update the dialog on this change.

GetNGraphPreview_OnDialogSetup

GetNGraphPreview_OnDestroy

GetNGraphPreview_OnCustomButton

See the **Creating Graph Preview GetN Dialog** section for an example of how to create an X-Function with a graph preview dialog.

19.2.4.5.5 GetNImageBox Event Handler Functions

The **event1** and **event1_ex** event handler functions are not used by X-Functions that have an image preview dialog. Instead you use the **GetNImageBox** functions.

GetNImageBox_OnInit

The event function called when the image GetN dialog is open. The function can decide whether to update the dialog, enable/disable the OK button or show an error message on the bottom of the dialog after calling.

GetNImageBox_OnChange

The event function called whenever any control on the dialog is modified by the user. This function can decide whether to update the dialog, update the image preview, enable/disable the OK button, or show an error message on the bottom of the dialog.

See the **Creating Image GetN Dialog** section for an example of how to create an X-Function with an image preview dialog.

19.2.4.6 Error Handling

19.2.4.6.1 Display an Error Message in the Dialog

When a user passes a bad value to your X-Function and also passes the **-d** option, or enters a bad value in the X-Function's dialog, you can display an error message in the dialog by setting the strErrMsg argument passed into the event handler function.

Which event handler function is called depends on the GetN Dialog your X-Function uses.

GetN Dialog Used	Event Handler Function
Simple GetNBox	xfname_event1
GetNGraphBox with Preview	GetNGraphPreview_OnChange
GetNImageBox	GetNImageBox_OnChange

Let us assume we have an X-Function that accepts an integer variable named **x1**. The following code snippet, inserted into the appropriate event handler function, checks the **x1** variable's value to see if it is less than zero or greater than 100. If it is then it sets the strErrMsg to an appropriate message to display in the dialog, and also disables the OK button. If the value is within the accepted range then it simply makes sure the OK button is enabled.

```
if( trGetN.x1.nVal < 0 || 100 < trGetN.x1.nVal )
{
    strErrMsg = "x1 is out of range.  Must be from zero to 100.";
```

```
    bOKEnable = false;

}

else

    bOKEnable = true;
```

19.2.4.6.2 Output Error Message

When an X-Function is being called programmatically and its dialog is not going to be shown, then you need another way to notify the user of any errors that may occur. Origin C has defined macros to output an error message to the Results Log, Command Window or Script Window. It is strongly recommended that you use these macros instead of out_str or printf.

The XF_WARN macros are used to output warning messages and do not terminate execution. All three macros accept a string as the first argument, containing the warning message. The two macros that accept additional arguments assume the warning message is also a format string for outputting one or two values.

XF_WARN(msg)

XF_WARN_EX(msg, arg1)

XF_WARN_EX2(msg, arg1, arg2)

```


// Assume the following:

// #define MIN_DATA_POINTS 5

// int nNumPts = 3;

// string strWksName = "Book1";

XF_WARN("Too few data points.");

XF_WARN_EX("Number of data points less than %d.", MIN_DATA_POINTS);

XF_WARN_EX("Worksheet named %s was not found.", strWksName);

XF_WARN_EX2("Only %d data points, need at least %d.", nNumPts, MIN_DATA_POINTS);
```

The XF_TRACE macro is similar to the XF_WARN macros in that it does not terminate execution. It also takes a message for the first argument, but it allows any number of additional arguments.

XF_TRACE(msg, ...)

```

```

```
// Assume the following:

// #define MIN_DATA_POINTS 5

// int nNumPts = 3;

// string strWksName = "Book1";

XF_TRACE("Only %d data points found in %s, need at least %d.",

    nNumPts, strWksName, MIN_DATA_POINTS);
```

The XF_THROW macros terminate execution, but everything else about them is the same as the XF_WARN macros. All three macros accept a string containing the warning message as the first argument. The two macros that accept additional arguments assume the warning message is also a format string for outputting one or two values.

XF_THROW(msg)

XF_THROW_EX(msg, arg1)

XF_THROW_EX2(msg, arg1, arg2)

```

// Assume the following:

// #define MIN_DATA_POINTS 5

// int nNumPts = 3;

// string strWksName = "Book1";

XF_THROW("Too few data points.");

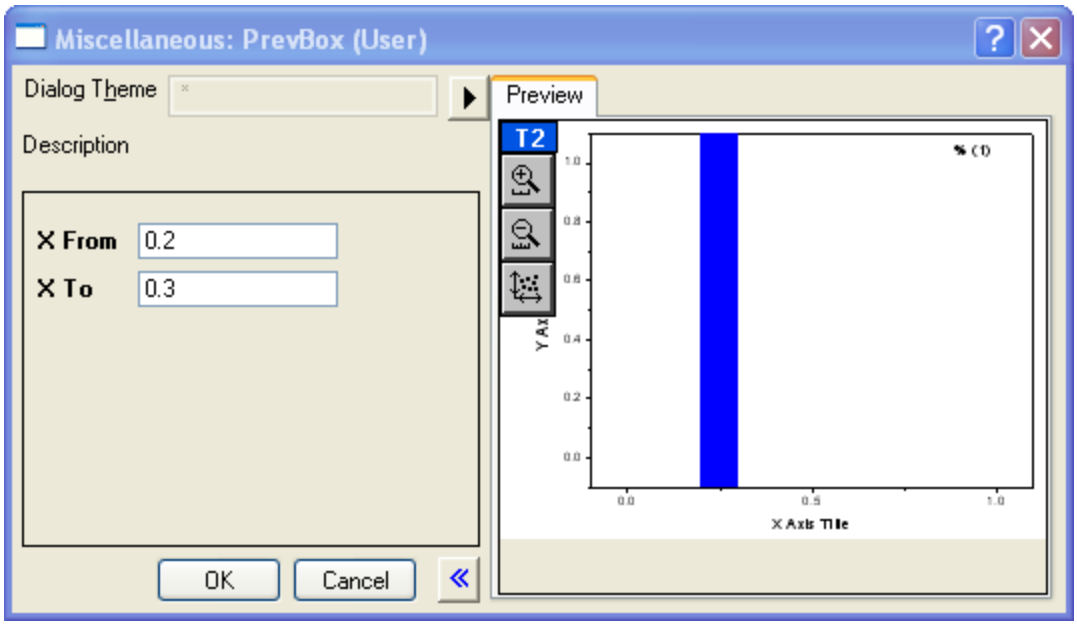
XF_THROW_EX("Number of data points less than %d.", MIN_DATA_POINTS);

XF_THROW_EX("Worksheet named %s was not found.", strWksName);

XF_THROW_EX2("Only %d data points, need at least %d.", nNumPts, MIN_DATA_POINTS);
```

19.2.5 Creating Graph Preview GetN Dialog

This section shows how to create an X-Function graph preview GetN dialog , including how to initialize a preview graph, how to update the preview graph on a GUI change, and how to update the GUI on the rectangle moving on the graph.

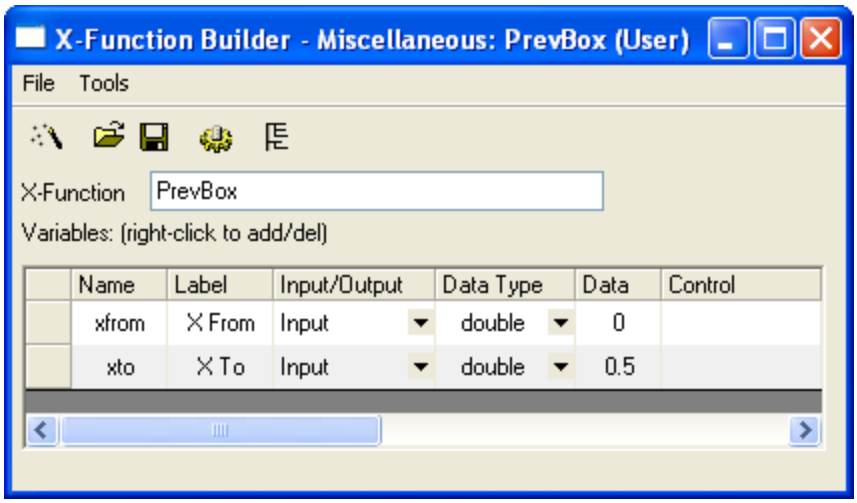


19.2.5.1 Create an Empty GetN Graph Preview Dialog

Press F10 to Open X-Function Builder.

Type "PrevBox" into the X-Function EditBox as the X-Function name.

Add a second variable by right-clicking on the Variables list and choosing "Add variable". Change the variable names, labels, and other values to match the settings in the dialog below.



Click Tree View button  in X-Function Builder, open the branch Usage Context -> Menu, set Auto GetN Dialog to GetNGraphBox with Preview, then click the Save button to save this X-Function.

Close X-Function Builder, and run "PrevBox -d" in Command window. A GetN graph preview dialog will show.

19.2.5.2 Coding X-Function in Code Builder

Open Origin Code Builder, in the menu choose File -> Open, locate the X-Function saved above, make sure the Add to Workspace check box is checked in the Open dialog, and click OK. PrevBox.XFC is added to the User folder in Origin C Workspace view and shows the function body in Edit view.

Add the additional header file after the line **//put additional include files here**. For example:

```
//put additional include files here
#include <..\Originlab\grobj_utils.h>
```

19.2.5.3 Initialize Preview Graph

The **GetNGraphPreview_OnInitGraph** function is used to initialize a preview graph. Copy the following codes and paste them into the function, then click the Compile button. When the preview dialog box opens, a rectangle will display on the preview graph. Click Compile and run "PrevBox -d" in the Command window.

```
// cast pgTemp from PageBase class object to GraphPage object
GraphPage gp(pgTemp);

// load this template to the graph in order to contain zoom
// in/out button on the graph. If this is not required,
// remove this line.
page_load(gp, "CurvePreview");

// get active graph layer in the graph page
GraphLayer gl = gp.Layers();

// add a rectangle, set x position reference to xfrom and xto
// variables
GraphObject go;
add_rect(gl, go, trGetN.xfrom.dVal, 0, trGetN.xto.dVal, -1,
    SYSCOLOR_BLUE, 2, LN_VERTICAL, true, false);
```

19.2.5.4 Update Preview Graph on GUI Change

To cause the preview graph to be updated, you need to include the UPDATE_GRAPH_CHANGED bit in the dwUpdateGraph variable in the **GetNGraphPreview_OnChange** function. For example:

```
// include the bit to call GetNGraphPreview_OnUpdateGraph
// to update graph
if( !(dwUpdateGraph & UPDATE_GRAPH_CHANGED) )
        dwUpdateGraph |= UPDATE_GRAPH_CHANGED;
```

Add the following code into the **GetNGraphPreview_OnUpdateGraph** function to update the graph. Click the Compile button and run "PrevBox -d" in the Command window.

```
// cast pgTemp from PageBase class object to GraphPage
// object
GraphPage gp(pgTemp);

GraphLayer gl = gp.Layers(); // get active graph layer
GraphObject go = gl.GraphObjects("Rect");

double x0 = trGetN.xfrom.dVal;
double x1 = trGetN.xto.dVal;

// update x position of rectangle on GUI
vector vv(4);
vv[0] = x0; vv[1] = x1; vv[2] = x1; vv[3] = x0;

Tree tr;
tr.Root.Data.X.dVals = vv;
```

```
go.UpdateThemeIDs(tr.Root);

go.ApplyFormat(tr, true, true);



// refresh graph window

gp.Refresh(TRUE);
```

19.2.5.5 Update GUI on Graph Object Event

If you want to update the GUI when you move/resize the rectangle on the preview graph, you need to:

Add the following to the **GetNGraphPreview_OnInitGraph** function after the **add_rect** function.

```


// set event script to notify preview dialog when rectangle

// is moved or resized. For other event types see the GRCT_*

// in oc_const.h file.

set_LT_script(go, "run -oc notify_dlg_on_move_obj;",

              GRCT_SIZEMOVE);
```

Add the following code to **GetNGraphPreview_OnChange** to update the new position of the rectangle to the GUI. Click the Compile button and run "PrevBox -d" in the Command window.

```


if( bChangedByGraph )

{

        // get rectangle x from/to and update GUI

        GraphPage gp(pgTemp);

        GraphLayer gl = gp.Layers();

        GraphObject goRect = gl.GraphObjects("Rect");



        Tree tr;

        tr = goRect.GetFormat(FPB_DATA, FOB_ALL, TRUE, TRUE);
```

```
        vector vv;

        vv = tr.Root.Data.X.dVals;



        trGetN.xfrom.dVal = vv[0];

        trGetN.xto.dVal = vv[1];



        // to include UPDATE_GRID_* bits to update GUI

        if( !(dwUpdateGrid & UPDATE_GRID_SIMPLE) )

                dwUpdateGrid |= UPDATE_GRID_SIMPLE;

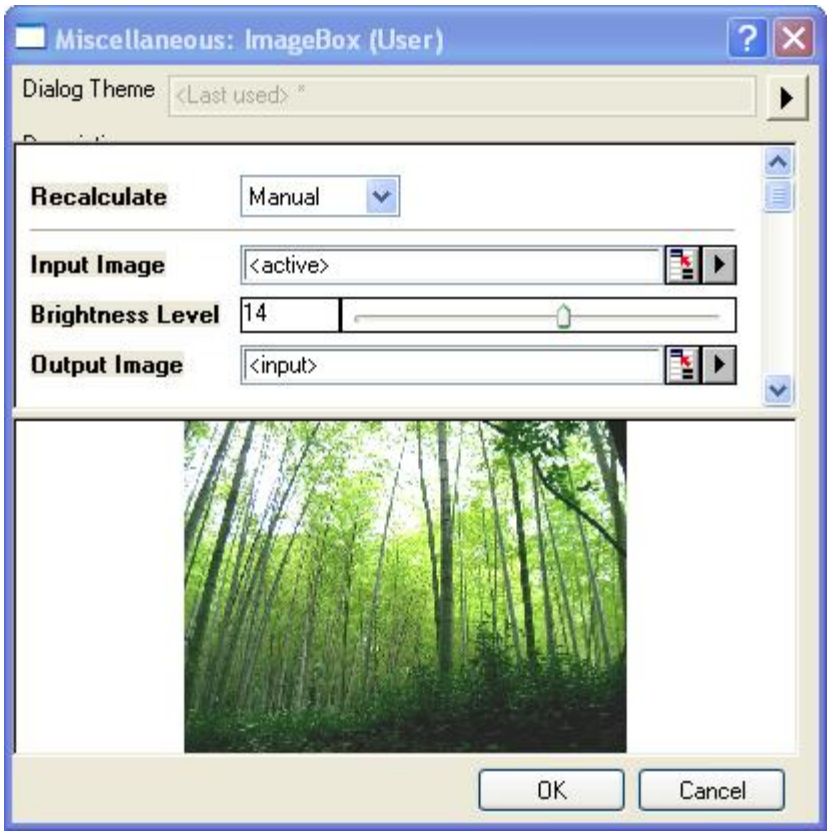
}
```

19.2.5.6 Add Preview buttons

To add a Preview button and an Auto Preview check box, please see X-Functions: Customizing an X-Function: Adding Custom Button on Dialog: Show Preview Button on Graph Preview GetN Dialog

19.2.6 Creating Image GetN Dialog

This section shows how to create an X-Function GetN dialog with a matrix image data preview as in the following dialog, including how to initialize an image preview and dialog controls, and how to update the image preview or dialog controls on any control change in the dialog performed by the user.

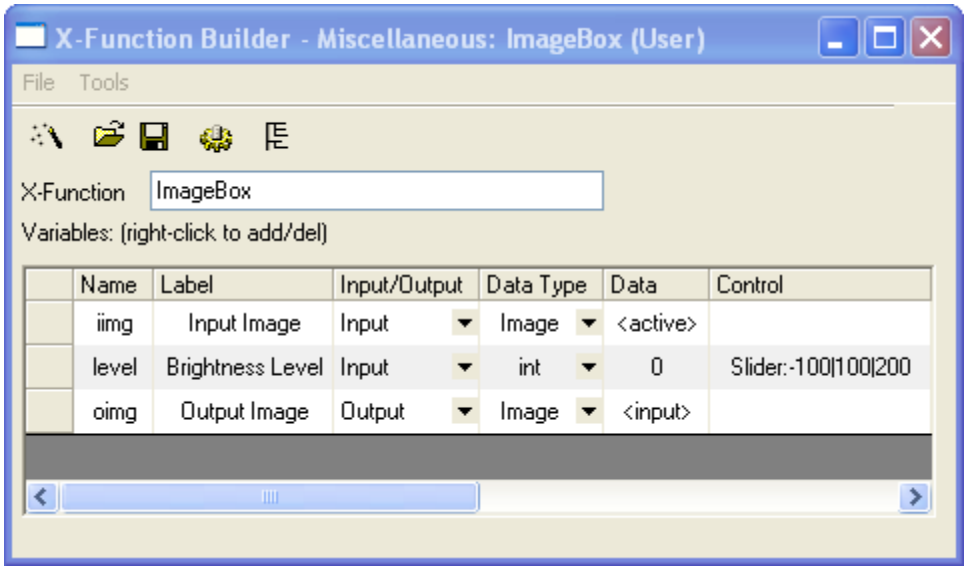


19.2.6.1 Create an Empty GetN Image Preview Dialog

Press F10 to Open X-Function Builder.

Type "ImageBox" into the X-Function EditBox for X-Function name.

Add more variables by right-clicking on the Variables list and choosing "Add variable". Change the variable names, labels, and other values to match the settings in the dialog below.



Click the Tree View button  on X-Function Builder, open the branch Usage Context -> Menu, set Auto GetN Dialog to GetNImageBox, then click the Save button to save this X-Function.

Close X-Function Builder, open a new matrix window, choose menu File -> Import -> Image, select an image file, and click Open to import the image into a matrix window.

Keep the matrix window active, and run "ImageBox -d" in the Command window. An image preview GetN dialog will show.

19.2.6.2 Coding X-Function in Code Builder

Open Origin Code Builder, in the menu choose File -> Open, locate the X-Function saved above, make sure the Add to Workspace check box is checked in the Open dialog, and click OK. ImageBox.XFC is added to the User folder in Origin C Workspace view and shows the function body in Edit view.

Add the additional header file after the line **//put additional include files here** as in the following.

```
//put additional include files here

#include <image_utils.h>

#include <..\Originlab\matdata_utils.h>
```

Add a static function, like the following, under the line **//put your own support static functions here**.

```
//put your own support static functions here

static bool _adjust_image_brightness_ex(BITMAPHANDLE* pPreviewBitmap,
TreeNode&

trGetN, bool& bOKEnable, string& strErrMsg)

{

        int nRet = adjust_image_brightness(pPreviewBitmap,
trGetN.level.nVal);



        if( SUCCESS != nRet)

        {

                bOKEnable = false; // disable OK button if adjust image
failed

                strErrMsg = "Fail to adjust image brightness"; // error
```

```
        message string

                return false;

        }

        return true;

}
```

19.2.6.3 GetNImageBox_OnInit Event Function

This event function is called when the GetN image preview dialog is open. It can be used to initialize an image preview, initialize dialog controls and enable/disable the OK button. For this example, add the following code into the GetNImageBox_OnInit function to adjust the image brightness according to the value of the Level control.

```
// Get bitmap handle pointer from image preview control
pBITMAPHANDLE pPreviewBitmap = pimgCntrl->GetBitmapHandlePtr();


// call internal function to adjust image
_adjust_image_brightness_ex(pPreviewBitmap, trGetN, bOKEnable, strErrMsg);
```

19.2.6.4 GetNImageBox_OnChange Event Function

This event function is called whenever any controls in the grid are modified by the user. It can be used to update the image preview on a dialog control change, update a dialog control on another control change, or update the OK button's enable/disable status. Add the following code into the GetNImageBox_OnChange function.

```
// call internal function to adjust image, return true to update image preview.
bUpdatePreview = _adjust_image_brightness_ex(pPreviewBitmap, trGetN,
                bOKEnable, strErrMsg);
```

19.2.6.5 Main Function

The main function has the same name as its corresponding X-Function. Add the following code into the main function and click the Compile button.

```
// Copy bitmap handle pointer from the input image to the output image
```

```
bool bCopy = true;

oimg.SetLBmp(iimg.GetLBmp(), bCopy);



// Get bitmap handle pointer from the output image

pBITMAPHANDLE phBmp = oimg.GetLBmp();



// adjust

adjust_image_brightness(phBmp, level);
```

19.2.6.6 Using X-Function

Run "ImageBox -d" in the Command window to open this X-Function dialog. Change the value of the Level control, and the brightness of the image preview will be changed.

Click the  button to open the theme context menu, choose Save As, type "mylevel" into the Theme Name box, then click OK to save the theme file. Click Cancel to close the X-Function dialog.

Run "ImageBox -d -t mylevel" in the Command window to open the X-Function dialog with the specified theme.

19.2.7 Creating Custom Dialog

X-Functions support custom dialog created in a dialog builder as interface. If the user wants to use the customized dialog, and also wants to support Auto Update in the dialog, or wants to open the dialog from the Origin menu, then they can create an X-Function with customized dialog. And then can create an OPX file to share it to others.

19.2.7.1 Creating a Dailog Resource DLL

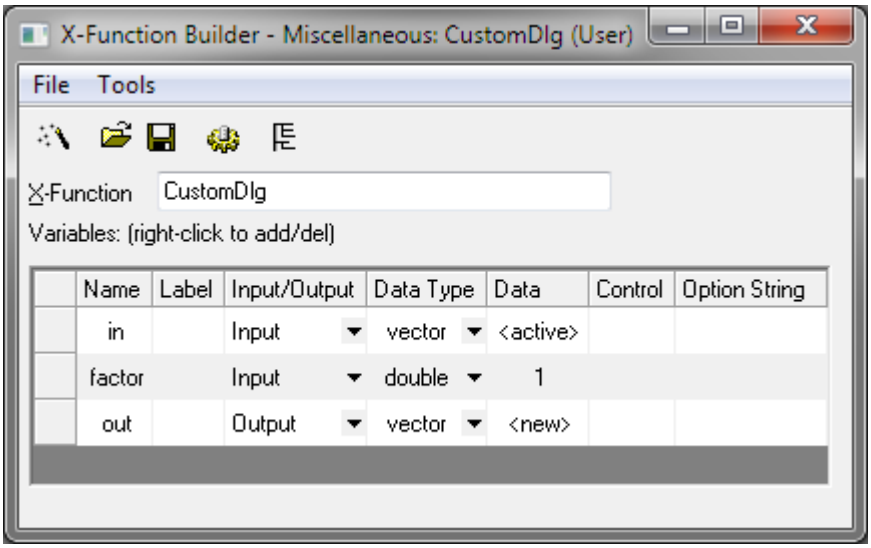
Here is an example to show you how to build a resource-only DLL in Visual Studio 2008 or 2010 which is accessible in Origin C. Follow the section **Creating a Resource-Only DLL**, we will get two files: **Welcome.dll** and **resource.h**.

19.2.7.2 Creating X-Function

Copy the above two files Welcome.dll and resource.h to Origin <User files Folder\OriginC\>. Note: the resource DLL file and resource.h file should be saved in the same folder.

Press F10 to open X-Function Builder.

Type "CustomDlg" as the X-Function name and set the X-Function variables as below:



Click the Tree View button to switch to Tree View, and choose Usage Context -> Menus -> Auto GetN Dialog to Custom. Then type the custom dialog function name, for example "OpenCustomDlg", into the Custom Dialog Function edit box. Click Save button to save this X-Function to the **default** folder under User Files folder and then close X-Function Builder. **Note**:X-Function should be save to the X-Functions folder under OriginExe folder or User Files Folder

Create a new header file named CustomDlg.h and save it to the "OriginC" folder. Type the following code into this file. The code in the CustomDialog construct function is used to show how to get the X-Function settings from the X-Function to the dialog, and how to set back the settings from the dialog to the X-Function by way of the Origin C TreeNode variable.

```
#include <..\Originlab\DialogEx.h>

#include "resource.h"

// file name of the resource without extension, here stand for Welcome.dll,

//and it assume the DLL is in the same folder as this .h file

#define STR_DLG_RESOURCE_DLL  "Welcome"


// the file name of resource DLL

#ifdef _OWIN64

#define STR_DLG_RESOURCE_DLL  "Welcome_64" // 64 bit DLL
```

```
#else //!_OWIN64

#define STR_DLG_RESOURCE_DLL  "Welcome" // 32 bit DLL

#endif// _OWIN64




class CustomDialog : public ResizeDialog

{

public:

    // Important: IDD_MY_DIALOG is the dialog resource ID.

    // STR_DLG_RESOURCE_DLL is the name of the dialog resource DLL file.

    CustomDialog(TreeNode& trXFSettings) : ResizeDialog( IDD_MY_DIALOG,
STR_DLG_RESOURCE_DLL )

    {

        m_trXFSettings = trXFSettings;



    }



    int DoModalEx(HWND hParent = NULL)

    {

        InitMsgMap();



        return ResizeDialog::DoModal(hParent);

    }



protected:

EVENTS_BEGIN

        ON_INIT(OnInitDialog)
```

```
        ON_OK(OnOK)
EVENTS_END


        BOOL OnInitDialog()
        {
                ResizeDialog::OnInitDialog();
                Edit    ed;
                ed = GetDlgItem(IDC_EDIT1);
                // init dialog control with xf setting
                ed.Text = ftoa(m_trXFSettings.factor.dVal);
                return TRUE;
        }


        BOOL OnOK()
        {
                Edit    ed;
                ed = GetDlgItem(IDC_EDIT1);
                m_trXFSettings.factor.dVal = atof(ed.Text);

                // If the input is 0, it will be changed to 1
automatically.
                if ( m_trXFSettings.factor.dVal == 0 )
                        m_trXFSettings.factor.dVal = 1;

                return TRUE;
        }
```

```
private:
        TreeNode        m_trXFSettings;
};
```

Open Origin Code Builder, in the menu choose File -> Open, select your X-Function, make sure the Add to Workspace check box is checked in the Open dialog, and click OK. CustomDlg.XFC is added to the User folder in Origin C Workspace view and shows the function body in Edit view.

Add the additional header file after the line **//put additional include files here**, as in the following.

```
//put additional include files here
#include "CustomDlg.h"
```

Add a function to open the custom dialog after the line **//put your own support static functions here**, as below. The function name is input into the Tree View Custom Dialog Function edit box. The function interface, return type, and argument list should remain the same, as in the following.

```
//put your own support static functions here
bool OpenCustomDlg(TreeNode& tr, PEVENT_FUNC pfn, LPCSTR lpcszTitle, LPCSTR
lpcszDescription, HWND hWndParent)
{
        CustomDialog dlg(tr);
        if( dlg.DoModal(GetWindow()) )
        {
                tr.AutoUpdate.nVal = AU_AUTO; // Default is 2(Manual)
                return true;
        }
        else // click Cancel button on dialog
        {
                return false; // will not call main function
        }
```

```
}
```

Put the following code into the X-Function's main function - CustomDlg function

```
out = in * factor;


// put comments like "Col(A)*8" on the Comments label of output column
DataRange drIn;
drIn = in.GetSourceDataRange();


Worksheet wks;
int c1, c2;
drIn.GetRange(wks, c1, c2);
Column col(wks, c1);


if( wks && col )
{
        string strOutputComment;
        strOutputComment.Format("Col(%s)*%g", col.GetName(), factor);
        out.SetOutputComment(strOutputComment);
}
```

Click 

button.

19.2.7.3 Using X-Function

Create a worksheet with two column and fill some different data.

Highlight column A and Run "CustomDlg -d" in the Command Window.

Fill "8" to Factor.

Click the OK button to close the dialog.

A new column will be created with an Auto Update lock (Column(A)*factor).

If the input is 0, it will be changed to 1 automatically.

Changing any data in column A will trigger Auto Update to update the output column.

19.2.7.4 How to share your custom dialog with others

You can use **Package Manager** to distribute it as a single package file to other Origin users. Then he can install your application by dropping the package file into Origin directly.

Select menu Tool->Package Manager. Click Add Files button to add the four files. CustomDlg.XFC, CustomDlg.h, resource.h, Welcome.dll.

Select File-> Save to save them as "CustomDlg.opx" file and share it with other users.

19.2.8 Creating Wizard Dialog

X-Functions support wizard dialogs. To create a wizard dialog, you need n+1 X-Functions, where n is the number of wizard pages. One is the main X-Function. The main X-Function is used to construct wizard pages and then open the wizard dialog if script mode is off, or run without dialog when script mode is on.

19.2.8.1 Creating a Wizard

The following shows how to create a two page wizard. Page 1 is used to select an XY range, and to specify the sub x range of the selected data range. Page 2 is used to add a rectangle with the specified fill color onto the source graph. The left x and right x of this rectangle are equal to the values of x from/to in Page 1.

Add a header file, wizDlg.h, under the UFF\OriginC\ folder. Put the following code into this header file to derive subclasses from Origin C built-in X-Function wizard classes. The X-Function basic wizard classes are:

XFCore, which is used to save and pass values in each page.

XFWizTheme, which is used to arrange theme settings for each wizard page.

XFWizInputOutputRange, which is used to arrange input and output ranges for each page.

XFWizManager, which is very important and must have a user-defined subclass derived from it. It is used to decide the steps in the wizard.

```
#ifndef __WIZ_DLG_H__
#define __WIZ_DLG_H__

#include <..\OriginLab\XFWiz.h>
```

```
#include <..\OriginLab\XFCore.h>


#include <..\OriginLab\WizOperation.h>
#include <..\OriginLab\XFWizNavigation.h>
#include <..\OriginLab\XFWizManager.h>
#include <..\OriginLab\XFWizScript.h>
#include <..\OriginLab\XFWizDlg.h>


#include <..\OriginLab\XFWizard_utils.h>


////////////////////////////////////////////////////////////////////////////
class WizDlgCore : public XFCore
{
public:
        WizDlgCore()
        {
        }


        void SetXRange(double from, double to)
        {
                m_dFrom = from;
                m_dTo = to;
        }


        void GetXRange(double& from, double& to)
        {
```

```
                from = m_dFrom;

                to = m_dTo;

        }


private:

        double m_dFrom;

        double m_dTo;

};


////////////////////////////////////////////////////////////////////////////

class WizDlgTheme : public XFWizTheme

{

};


////////////////////////////////////////////////////////////////////////////

class WizDlgInputOutputRange : public XFWizInputOutputRange

{

};


////////////////////////////////////////////////////////////////////////////

class WizDlgManager : public XFWizManager

{

public:

        WizDlgManager(LPCSTR lpcszThemeName = NULL,

        const XFWizTheme *pXFWizTheme = NULL,

        const XFWizInputOutputRange *pXFWizIO = NULL, DWORD dwUIDOp = 0);
```

```
protected:
        virtual XFCore*       CreateXFCore() { return new WizDlgCore; }
        virtual XFWizTheme*   CreateXFWizTheme() { return new WizDlgTheme;
}
        virtual XFWizInputOutputRange* CreateXFWizInputOutputRange()
        {
                return new WizDlgInputOutputRange;
        }

        virtual string     GetClassName() { return "WizDlgExample"; }
};

#define STR_XFNAME_PAGE_1                   "wizPage1"
#define STR_XFNAME_PAGE_2                   "wizPage2"

#define STR_LABEL_XFNAME_PAGE_1             "Page 1"
#define STR_LABEL_XFNAME_PAGE_2             "Page 2"

WizDlgManager::WizDlgManager(LPCSTR lpcszThemeName,
        const XFWizTheme *pXFWizTheme, const XFWizInputOutputRange
*pXFWizIO,
        DWORD dwUIDOp) // = NULL, NULL, NULL, 0
: XFWizManager(lpcszThemeName, pXFWizTheme, pXFWizIO, dwUIDOp)
{
        StringArray saMapXFNames = {STR_XFNAME_PAGE_1, STR_XFNAME_PAGE_2};
        StringArray saMapXFLabels = {STR_LABEL_XFNAME_PAGE_1,
```

```
                                    STR_LABEL_XFNAME_PAGE_2};

        m_saMapXFNames = saMapXFNames;

        m_saMapXFLabels = saMapXFLabels;

        ASSERT( m_saMapXFNames.GetSize() == m_saMapXFLabels.GetSize() );


        m_saDefaultXFNames = saMapXFNames;

}


////////////////////////////////////////////////////////////////////////////

int run_wiz_nodlg(LPCSTR lpcszThemeName = NULL,

        const XFWizTheme *pXFWizTheme = NULL,

        const XFWizInputOutputRange *pXFWizIO = NULL, DWORD dwOPUID = 0)

{

        TEMPLATE_run_wiz_nodlg(WizDlgManager, lpcszThemeName, pXFWizTheme,

                        pXFWizIO, dwOPUID)

}


int open_wiz_dlg(LPCSTR lpcszThemeName = NULL,

        const XFWizTheme *pXFWizTheme = NULL,

        const XFWizInputOutputRange *pXFWizIO = NULL, DWORD dwOPUID = 0)

{

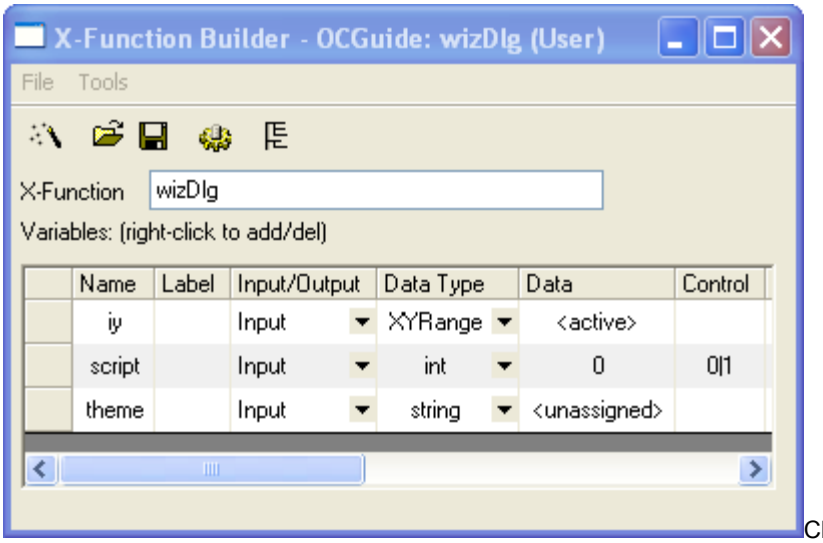
        TEMPLATE_open_wiz_dlg(WizDlgManager, lpcszThemeName, pXFWizTheme,

                        pXFWizIO, dwOPUID)

}


#endif     //__WIZ_DLG_H__
```

Press F10 to open X-Function Builder, to create a main X-Function as below.



Click the  button to save this X-Function. Close X-Function Builder.

Open Code Builder, open the wizDlg.OXF file and add it to the current workspace. Add the following header files after the line **//put additional include files here**.

```
#include <..\OriginLab\XFWiz.h>

#include <event_utils.h>
#include <..\OriginLab\WksOperation.h>
#include <..\OriginLab\WizOperation.h>

#include <..\OriginLab\XFCore.h>
#include <..\OriginLab\XFWizNavigation.h>
#include <..\OriginLab\XFWizManager.h>
#include <..\OriginLab\XFWizScript.h>
#include <..\OriginLab\XFWizDlg.h>
#include <..\OriginLab\XFWizard_utils.h>

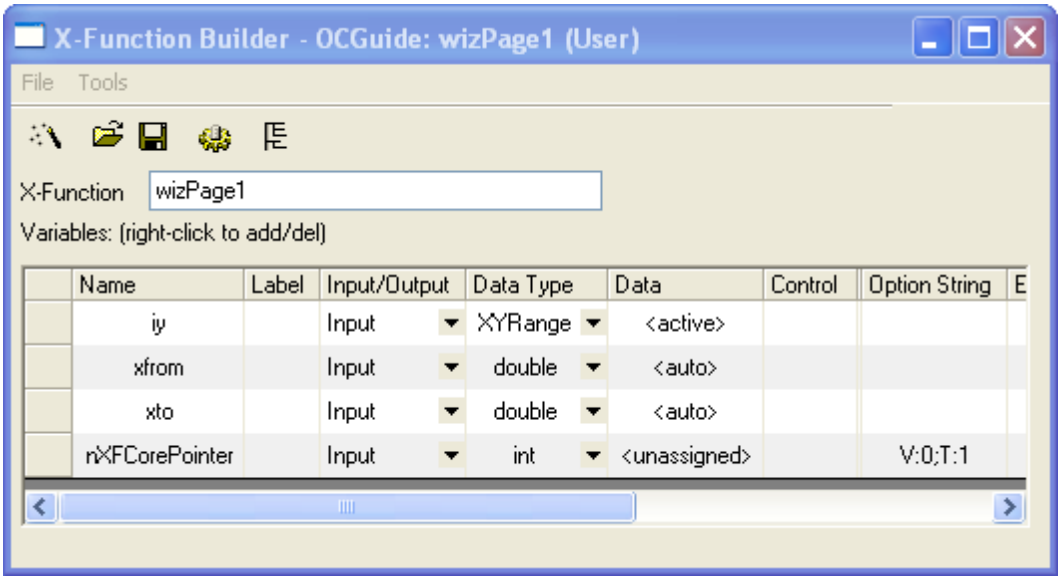
#include <..\Originlab\wizDlg.h> // user added header file for this example
```

Add the following code to the main function **wizDlg**

```
WizDlgInputOutputRange objXFWizIO;
if(!xfwiz_construct_input_range(&objXFWizIO, STR_XFNAME_PAGE_1,  iy))
{
        XF_THROW(XFERR_FAIL_TO_UPDATE_GETN_TREE);
}

if( script )
{
        run_wiz_nodlg(theme, NULL, &objXFWizIO);
}
else
{
        open_wiz_dlg(theme, NULL, &objXFWizIO);
}
```

Press F10 to open X-Function Builder and create the X-Function wizPage1 for the wizard page 1 as below.



Add the following user-defined header file after the line **//put additional include files here**, as below.

```
#include <..\Originlab\wizDlg.h> // user added header file for this example
```

Put the following code into the page 1 main function **wizPage1**.

```
WizDlgCore* pWizCore =
        (WizDlgCore*)get_xf_core_handler(NODE_NAME_XF_WIZ_CORE_POINTER);
ASSERT(pWizCore);

// Set X From/To to WizCore class object
pWizCore->SetXRange(xfrom, xto);
```

Add the following code to the wizPage1 X-Function _event1 function **wizPage1_before_execute**, to initialize the xfrom and xto controls according to the input data range.

```
if( PDS_AUTO == okutil_cvt_str_to_predefined_type(trGetN.xfrom.strVal)
        || PDS_AUTO == okutil_cvt_str_to_predefined_type(trGetN.xto.strVal)
)
{
        DataRange dr;
        TreeNode trRange = trGetN.iy;
        okxf_resolve_tree_construct_range(&trRange, &dr);

        vector vx;
        dr.GetData(DRR_GET_DEPENDENT, 0, NULL, NULL, NULL, &vx);

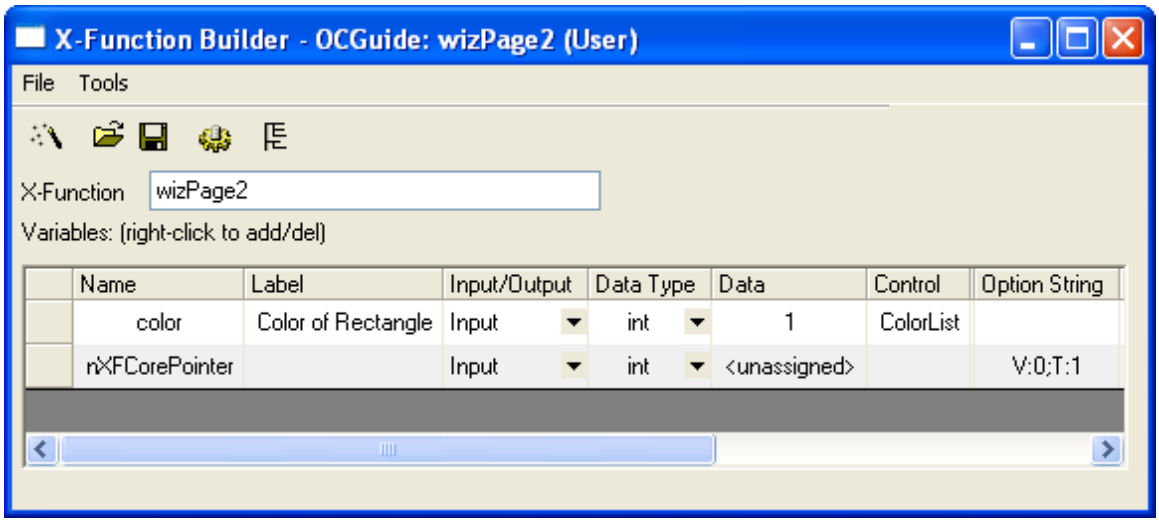
        double from, to;
        vx.GetMinMax(from, to);
```

```
        trGetN.xfrom.dVal = from;

        trGetN.xto.dVal = to;

}
```

Open X-Function Builder to create the X-Function wizPage2 for page 2. Set up the X-Function name and variables as below and click the Save button to save.



Open the wizPage2.OXF file in Code Builder, and add it to the current workspace to include header files, as below.

```
//put additional include files here
#include <..\Originlab\wizDlg.h>
#include <..\Originlab\grobj_utils.h>
```

Find the main function **wizPage2** and put the following code into it. The code is used to add a rectangle according to the xfrom and xto values assigned in page 1.

```
WizDlgCore* pWizCore =
        (WizDlgCore*)get_xf_core_handler(NODE_NAME_XF_WIZ_CORE_POINTER);
ASSERT(pWizCore);

// Get the x from/to values that are specified in Page 1
```

```
double xfrom, xto;
pWizCore->GetXRange(xfrom, xto);


// To get input range.
XFWizNavigation* pXFWizNavg =
        (XFWizNavigation*)(pWizCore->GetXFWizNavigation());
WizDlgInputOutputRange* pwizInputOutputRange =
        (WizDlgInputOutputRange*)(pXFWizNavg->GetXFWizInputOutputRange());


// Get input ranges from Page 1
Array<DataRange&> arrdr;
if( pwizInputOutputRange->Get(&arrdr, STR_XFNAME_PAGE_1, true) )
{
        DataRange& dr = arrdr.GetAt(0);


        vector<uint> vUIDs;
        DataPlot dp;
        if( dr.GetPlots(vUIDs) > 0 )
        {
                // Get source graph layer from input range
                dp = (DataPlot)Project.GetObject(vUIDs[0]);
                GraphLayer gl;
                dp.GetParent(gl);


                // To add rectangle to graph
                GraphObject go;
```

```
                add_rect(gl, go, xfrom, 0, xto, 0, color, 2, LN_VERTICAL,
                        true, false);
            }
        }
```

19.2.8.2 Using X-Function

Before running the example wizard, you need to create a new worksheet, fill two columns with data, and plot a line graph with two columns.

19.2.8.2.1 Run with wizard dialog

Keep the graph window active. Run "wizDlg" in the Command window.

In the dialog that appears, change the values of the xfrom and xto controls. Click the Next button.

In the next page, change the Fill Color control to Blue.

Click the theme button , choose Save As, type in theme1 as the name, and click OK.

Click the Finish button. The wizard dialog closes and a blue rectangle will be added to your source graph.

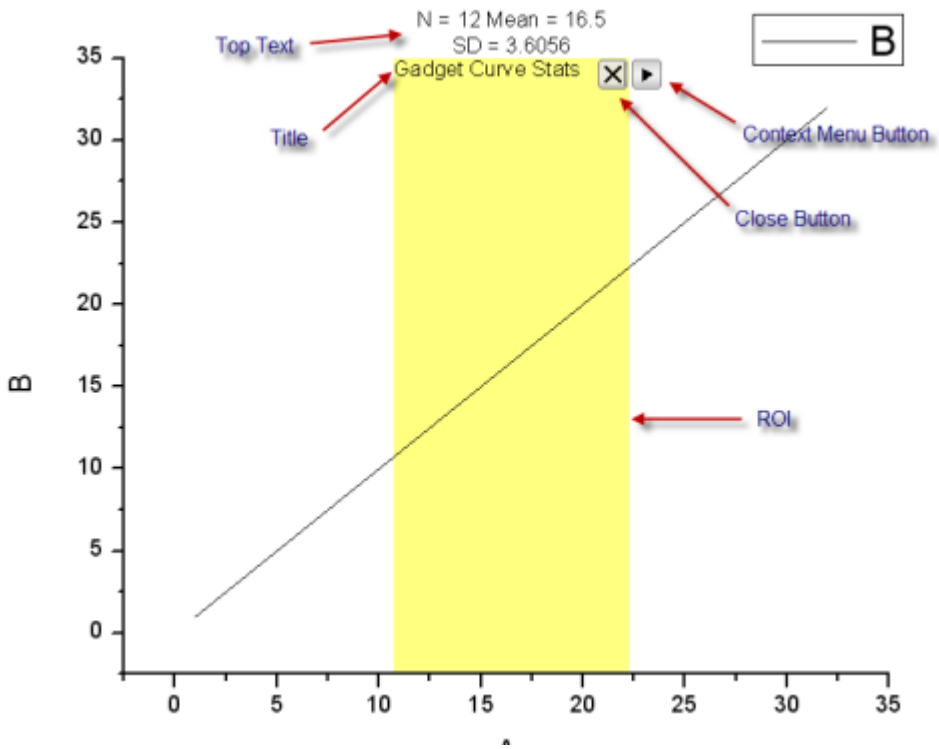
19.2.8.2.2 Run with script mode without dialog

Keep the graph window active. Run "wizDlg s:=1 t:=theme1" in the Command window. No dialog opens and a rectangle will be added to the active graph with the settings specified in the theme file.

19.2.9 Creating a Gadget Tool with X-Function

Origin offers some gadgets that operate on graphs and matrices. You can find them in the Gadgets menu when a graph or matrix is active.

When you choose one of the gadgets, Origin adds a rectangle to the graph or matrix, allowing you to choose the region of data to be analyzed. After selecting the region of interest (ROI), you can open the context menu by clicking the button and using the preference dialogs or themes to set up the analysis and output the results to desired destinations.



Origin allows users to add a user-defined gadget tool with an X-Function, and then add this tool to the Origin Gadgets menu.

19.2.9.1 How to Create a Gadget

The following are the three main steps for creating a gadget tool:

Create an X-Function.

The X-Function handles the dialog theme and adds the gadget tool to Origin's Gadget menu.

Create an Origin C class derived from GraphObjCurveTool or MatObjROITool.

The GraphObjCurveTool and MatObjROITool classes are Origin C building classes and declared in the file: *OriginExeFolder*\OriginC\Originlab\GraphObjTools.h

These classes are used to create a region of interest (ROI) object and provide default handling for events. Events include moving and resizing the ROI object and choosing items on the ROI context menu.

Add event functions.

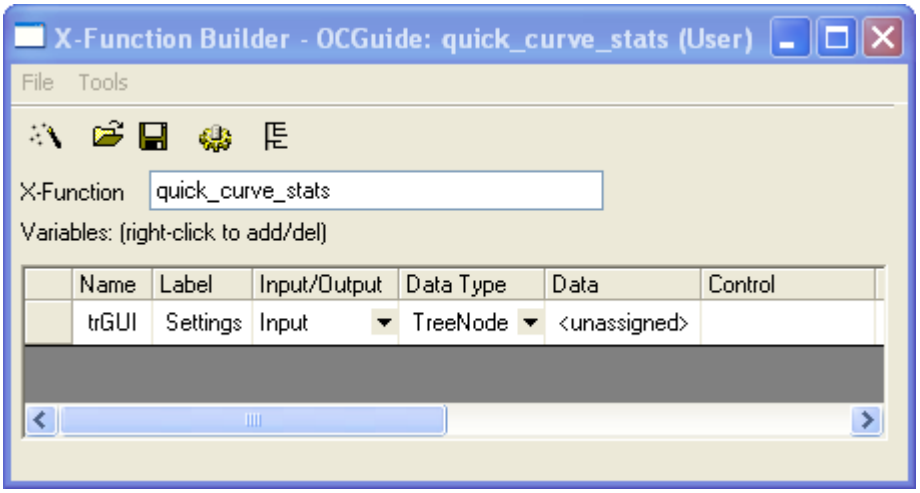
The name of the global event function needs to be in the *xfname*_events format, where *xfname* is the name of the X-Function. A function's argument list must use the following syntax:

```
void xfname_events(string strGrName, int nEvent, int nMsg = 0);
```

The next sections will explain these steps in more detail.

19.2.9.2 Creating an X-Function

With Origin active, press F10 to open X-Function Builder. Set up your new X-Function's name and variables as shown below.



Click the  button to save the X-Function.

Click the  button to open the X-Function's source code in Code Builder.

Add two additional include statements.

```
//put additional include files here
#include <..\Originlab\grobj_utils.h>
#include <..\Originlab\GraphObjTools.h>
```

Find the function named **quick_curve_stats_make_tree** and replace it's body with the following code for creating a GUI tree.

```
if( 0 == strcmp(lpcszVarName, "trGUI") )
{
    GETN_USE(tr)
    GETN_OPTION_BRANCH(GETNBRANCH_OPEN) // default to open branch

    int nUserID = GET_USER_DATAID(0);
```

```
    int nBranchID = 1;



    GETN_STR(toolname, STR_TOOLNAME, "Quick Curve Stats")GETN_ID(nUserID++)

        // the option to add a checkbox control with editbox

        GETN_CONTROL_OPTION_BOX(1)

    GETN_COLOR(rectColor, "Rectangle Fill Color", SYSCOLOR_LTYELLOW)
GETN_ID(nUserID++)

        GETN_COLOR_CHOICE_OPTIONS(COLORLIST_CUSTOM | COLORLIST_SINGLE)



    // quantities branch, for output result options

    GETN_BEGIN_BRANCH(quantities, "Quantities") GETN_ID(nBranchID++)

        GETN_OPTION_BRANCH(GETNBRANCH_OPEN|GETNBRANCH_CHECK_CONTROL)



        GETN_CHECK(dataname, "Dataset Name", true) GETN_ID(nUserID++)

        GETN_CHECK(n, "Total Number", true) GETN_ID(nUserID++)

        GETN_CHECK(sum, "Sum", true) GETN_ID(nUserID++)

        GETN_CHECK(mean, "Mean", true) GETN_ID(nUserID++)

        GETN_CHECK(sd, "Standard Deviation", false) GETN_ID(nUserID++)

        GETN_CHECK(se, "SE of Mean", false) GETN_ID(nUserID++)

    GETN_END_BRANCH(quantities) // the end of quantities branch



    // output branch, the options for output destination

    GETN_BEGIN_BRANCH(output, "Output To") GETN_ID(nBranchID++)

        GETN_CHECK(script, "Script Window", true) GETN_ID(nUserID++)

        GETN_CHECK(reslog, "Results Log", false) GETN_ID(nUserID++)

        GETN_CHECK(outputwks, "Output to Worksheet", false)
GETN_ID(nUserID++)
```

```
    GETN_END_BRANCH(output) // the end of output branch

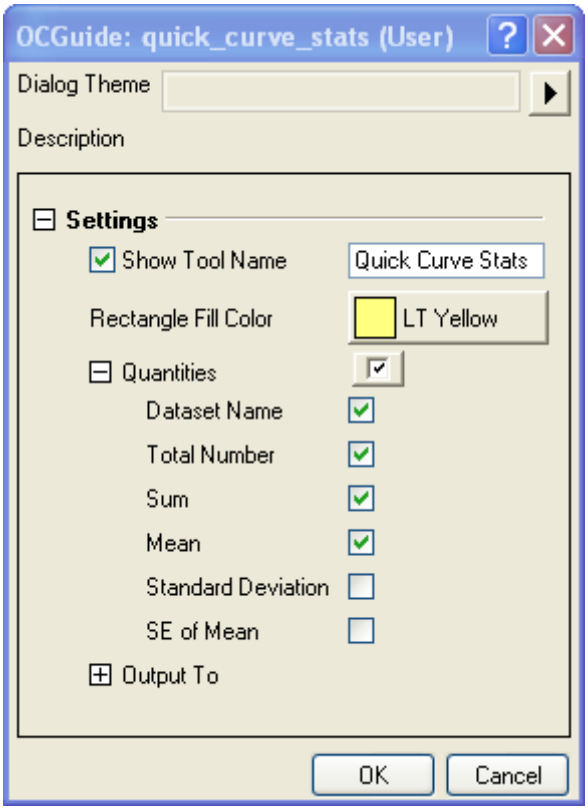
}

return 0;
```

Click the Compile button to compile your changes. After successfully compiling, click the Return to Dialog button to go back to X-Function Builder.

Click the button to save the X-Function again, then close X-Function Builder.

Run "quick_curve_stats -d" in the Command window. The following dialog will display.



Now you have your X-Function. In the next section we will show you how to define a new class derived from an Origin C building class.

19.2.9.3 Defining a Class Derived from GraphObjCurveTool

This section will show you how to define a new class derived from Origin C's GraphObjCurveTool class. The GraphObjCurveTool class will help handle most of the work for your new class.

The GraphObjCurveTool class is used when making a gadget that will operate on a graph. If you are making a gadget that will operate on matrix data then you would derive your class from the MatObjROITool class. Both GraphObjCurveTool and MatObjROITool are declared in the GraphObjTools.h header file.

Open quick_curve_stats.XFC in Code Builder. Near the top of the file we will define our new class.

```
//put additional include files here
#include <GetNbox.h>
#include <..\originlab\grobj_utils.h>
#include <..\Originlab\GraphObjTools.h>

#define XF_NAME "quick_curve_stats"
#define TOOL_PREFERENCES_TITLE _L("Statistics")

class QuickCurveStatsTool : public GraphObjCurveTool
{
protected:
        string  GetXFName() { return XF_NAME; }
        string  GetSignature() { return "xf_addtool_quick_curve_stats"; }
        string  GetPreferenceTitle(){ return TOOL_PREFERENCES_TITLE; }

        // Will rewrite this function in Adding Event Functions section
below.
        bool DoOutput(bool bUpdateLastOutput, bool bMarkerOnly = false)
        {
                out_str("DoOutput");
                return true;
        }
};
```

Go to the X-Function's main function, and use the class defined above to create an ROI on the graph.

```
QuickCurveStatsTool curveTool();

int nErr = curveTool.Create(XF_NAME, LN_VERTICAL, trGUI);

if(nErr)

{

    error_report("quick_curve_stats failed to Init!");

    return;

}

curveTool.ExpandFullRange();
```

Activate a graph window with a curve, and run "quick_curve_stats -d" in the Command window. A rectangle with top text, a title and two mini buttons will be added on the graph.

19.2.9.4 Adding Event Functions

19.2.9.4.1 Basic global event function

Open quick_curve_stats.XFC in Code Builder, and add the event function as below, after the line **//put your own support static functions here**

```
//put your own support static functions here

void quick_curve_stats_events(string strGrName, int nEvent, int nMsg = 0)

{

        QuickCurveStatsTool sTool;

        graphobjtool_events(sTool, strGrName, nEvent, nMsg);

}
```

Run the X-Function, "quick_curve_stats -d", in the Command window and click the OK button to close the dialog. Click the  button, attached to the ROI, to open the context menu. Click New Output, and DoOutput will be printed on the Script window. Choose Preference, and a dialog with the X-Function's settings will appear. Change the color in the Rectangle Fill Color control, and update the text in the Show Tool Name control in the opening dialog, then click OK. The fill color and title text of the rectangle will be changed.

19.2.9.4.2 Overload virtual event functions

Run this X-Function, "quick_curve_stats -d", in the Command window to add a ROI. If you move the ROI on a graph, the Origin status bar on the left will show the current x from/to values and the width of the ROI. The

following three steps will show you how to overload the OnMove virtual method to refresh the ROI top text with the curve statistics results, on 'ROI moving'.

Declare the following two methods in the QuickCurveStatsTool class.

```
protected:
        BOOL OnMove()
        {
                string strTopText = getResult();
                UpdateTopLabel(strTopText, true);
                return true;
        }
private:
        string getResult(TreeNode& trResult = NULL)
        {
                // Get the data in ROI selected
                vector vx, vy;
                GetData(vx, vy);

                // Calculate statistic on data
                int npts;
                double sum, mean, sd, se;
                vy.Sum(sum);
                if( 0 != ocmath_basic_summary_stats(vy.GetSize(), vy,
&npts,
                        &mean, &sd, &se) )
                {
                        return "Error in calculation!";
```

```
                }


                // Get output options from GUI
                Tree trGUI;
                GetGUITree(trGUI);
                bool       bDatasetName = trGUI.quantities.dataname.nVal,
                                bPoints = trGUI.quantities.n.nVal,
                                bSum = trGUI.quantities.sum.nVal,
                                bMean = trGUI.quantities.mean.nVal,
                                bSD = trGUI.quantities.sd.nVal,
                                bSE = trGUI.quantities.se.nVal;


                string strDatasetName;
                if( bDatasetName )
                {
                        m_dp.GetRangeString(strDatasetName);
                }


                // Put result to string
                string strText;
                string strNumDec = "*", strNextLine = "\r\n";
                if( bDatasetName )
                        strText += "Dataset Name: " + strDatasetName +
strNextLine;
                if( bPoints )
                        strText += "N = " + npts + strNextLine;
                if( bSum )
```

```
                        strText += "Sum = " + ftoa(sum, strNumDec) +
strNextLine;
                if( bMean )
                        strText += "Mean = " + ftoa(mean, strNumDec) +
strNextLine;
                if( bSD )
                        strText += "SD = " + ftoa(sd, strNumDec) +
strNextLine;
                if( bSE )
                        strText += "SE = " + ftoa(se, strNumDec) +
strNextLine;
                strText.TrimRight(strNextLine);


                // Put result to tree
                if( trResult && trResult.IsValid() )
                {
                        GETN_USE(trResult)
                        if( bDatasetName )
                        {
                                GETN_STR(DatasetName, "Dataset Name",
strDatasetName)

GETN_CURRENT_SUBNODE.SetAttribute(STR_TYPE_ATTRIB,
                                        TNVAL_TYPE_CSTRING);
                        }
                        if( bPoints )
                        {
                                GETN_NUM(N, "Number of Points", npts)
                        }
                        if( bSum )
```

```
                {
                        GETN_NUM(Sum, "Sum", sum)
                }
                if( bMean )
                {
                        GETN_NUM(Mean, "Mean", mean)
                }
                if( bSD )
                {
                        GETN_NUM(SD, "SD", sd)
                }
                if( bSE )
                {
                        GETN_NUM(SE, "SE", se)
                }
        }

        return strText;
}
```

Run the X-Function to add a ROI on your graph. Move the ROI, and the top text will be updated with the current statistics results of the selected Y data.

The following shows how to make the context menu items "New Output" and "Go to Report Worksheet" work. You need to overload the DoOutput and GetReportWorksheet virtual functions in the derived class QuickCurveStatsTool, and add a private function, acessBinaryInfo, to set and get the report worksheet name in the rectangle's binary storage. If you fail to get the report worksheet name from the rectangle, or the report worksheet name represents an invalid worksheet, the "Go to Report Worksheet" context menu item will be gray. Add the following methods to the QuickCurveStatsTool class.

```
protected:
        bool DoOutput(bool bUpdateLastOutput)
        {
                Tree trGUI;
                GetGUITree(trGUI);

                // Get output result to string and treenode
                Tree tr;
                TreeNode trResult;
                if( trGUI.output.outputwks.nVal )
                        trResult = tr.AddNode("Result");
                string strResult = getResult(trResult);

                // Output to Script window, optional
                if( trGUI.output.script.nVal )
                {
                        LT_execute("type -a"); // to open Script window if
closed
                        out_str(strResult);
                }

                // Output to Result Log, optional
                if( trGUI.output.reslog.nVal )
                {
                        Project.OutStringToResultsLog(strResult);
                }
```

```
                // Output to worksheet, optional
                if( trGUI.output.outputwks.nVal )
                {
                        // To check output worksheet if already existed
                        Worksheet wks;
                        string strOutputWks;
                        bool bCreateNew = true;
                        if( acessBinaryInfo(true, strOutputWks) )
                        {
                                wks.Attach(strOutputWks);
                                bCreateNew = !wks.IsValid();
                        }

                        // If it doesn't exist, you need to create a new
one,
                        // then put worksheet name into rectangle binary
storage
                        if( bCreateNew )
                        {
                                wks.Create(NULL, CREATE_HIDDEN);
                                wks.SetSize(-1, 0);

                                wks.GetRangeString(strOutputWks);
                                acessBinaryInfo(false, strOutputWks);
                        }
                        out_tree_to_wks(trResult, wks);
                }
```

```
                return true;
        }

        BOOL GetReportWorksheet(Worksheet& wksReport)
        {
                string strWks;
                if( acessBinaryInfo(true, strWks) )
                {
                        wksReport.Attach(strWks);
                        return wksReport.IsValid();
                }
                return false;
        }

private:
        bool acessBinaryInfo(bool bGet, string& strInfo)
        {
                string strStorageName = "Info";
                Tree tr;

                if( bGet )
                {
                        // To get output worksheet name from rectangle
binary storage
                        if( m_go.GetBinaryStorage(strStorageName, tr)
                                && tr.OutputWks
                                && !tr.OutputWks.IsEmpty() )
```

```
                        {
                                strInfo = tr.OutputWks.strVal;
                                return true;
                        }
                        return false;
                }
                else
                {
                        // Save output worksheet name to rectangle binary
storage
                        tr.OutputWks.strVal = strInfo;
                        return m_go.PutBinaryStorage(strStorageName, tr);
                }
        }
```

Prepare a graph with a data plot set as active, and run "quick_curve_stats -d" in the Command window to add a ROI. Move the ROI to an interesting area on the graph, open the context menu, and choose Preferences to open a dialog. Select all the check boxes in the Quantities and Output To branch, and click OK to close the dialog. Open the context menu again, and notice the Go to Report Worksheet menu item is gray. Click the New Output menu, and the result string will be outputted to a Script window and a Result Log, and a hidden worksheet will be created. Choose the Go to Report Worksheet menu, and the report worksheet displays. The columns in the worksheet hold the results.

19.3 Customizing an X-Function

19.3.1 Customizing an X-Function

This section covers the following topics:

Putting an X-Function in the Product Menu

Adding Documentation to an X-Function

Adding Custom Button on Dialog

19.3.2 Putting an X-Function in the Product Menu

Putting an X-Function into the Origin menu involves the following two steps:

Modifying basic settings in the X-Function Builder Tree View

Putting the X-Function into the specified main menu and sub menu

Then, in the menu, choose Preference: Menu:Full Menus, or restart Origin to auto-scan the changes for all X-Functions and add the newly added X-Function to the menu. Please refer to X-Function Builder's documentation for more details.

19.3.2.1 Setup in Tree View

To make your X-Function show properly in the menu, the following settings in X-Function Builder are required:

Open the X-Function in X-Function Builder.

Switch to Tree View mode, in the Usage Context branch.

Choose **Main Menus** for Menus.

Check the **Windows** check box.

Check one or several window types: Worksheet, Graph, or Matrix.

19.3.2.2 Menu Accessible

Once the X-Function is set up in Tree View, you can put the X-Function into the specified menu. Origin enables the user's X-Function to be accessible from the two kinds of menus:

Default menu

(This mechanism has a few limitations: you need to put the X-Function into the specified file folder, and the X-Function name needs to follow the specified prefix. Also, it is not easy to control its location in the menu. The next mechanism, a user defined menu, will solve all these problems.)

User defined menu

19.3.2.2.1 Default Menu

X-Functions are accessible from a specified default menu if they are saved in the proper sub-folder of the **X-Functions** folder under the User folder.

All X-Functions in the subfolders below will be shown in the menu.

Folder	Menu
Signal Processing\	Analysis: Signal Processing
Spectroscopy\	Analysis: Spectroscopy
Mathematics\	Analysis: Mathematics
Statistics\	Statistics

All X-Functions with the specified prefix in the subfolders below will be shown in the menu.

Folder	Prefix	Menu
Import and Export\	imp*	Data: Import from File
	exp*	File: Export
Data Manipulation\	w*	Worksheet
	col*	Column
Database Access\	db*	File: Database Access

The X-Functions in the Miscellaneous, Data Exploration, and Utilities folders will never be shown in the menu, except if the user directly modifies the XML file, the process for which will be described in the next section.

19.3.2.2.2 User Defined Menu

In addition to adding X-Functions to default menus, the user is able to put X-Functions into user-defined menus by editing the XML files under the OriginExePath\Themes\Menus folder. There are two XML files under this folder: ConfMenus.xml is used to control the Origin main menu, and ContextMenus.xml is for the context menu.

Example

The following is a complete example that shows how to add a user-defined X function to a new sub menu, **OC Guide Tools**, under Origin's **Analysis** main menu. The user-defined menu is the mid in **Analysis:Fitting** and **Analysis:Signal Processing**.

Create a folder **OC Guide** under the path *<User Files Folder>\X-Functions*\.

Create a user-defined X function following this example, but name it as **MyAnalysis** and make sure it is saved under the path *<User Files Folder>\X-Functions\OC Guide*\.

Open the **MyAnalysis** X-Function in X-Function Builder. Switch to Tree View, in the Usage Context branch, set Menus as Main Menus, check the Windows check box, and check the Worksheet check box under the Windows branch.

Type some text into the Description edit box, for example, "&Custom Dialog". & is used to access the quick selection menu with the keyboard. Click Save to save changes to the X-Function and close X-Function Builder.

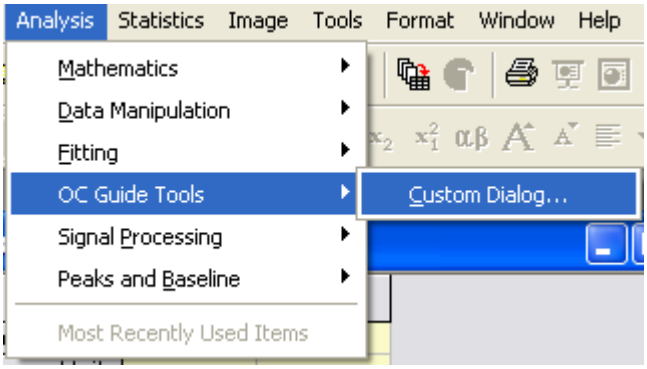
Open **ConfMenus.xml** in an XML editor, for example, XML Marker.

Choose **wks -> wksAnalysis -> wfit** in the left panel.

Copy the following and paste it into the right panel after <wfit...></wfit>, then click Save.

```
<testmenu1 Label = "OC Guide Tools" popup = "1">
        <efg Label = "">
            <name>*</name>
            <auxopts>2162688.</auxopts>
            <catsubcat DataID = "74">OC Guide</catsubcat>
        </efg>
</testmenu1>
```

Close Origin and restart. Activate a worksheet and, in the menu, choose Analysis. Menu Test:Custom Dialog will display.



19.3.3 Adding Documentation to an X-Function

Documentation is necessary for an X-Function to be complete. In the documentation, the X-Function creator needs to tell the user something like how to use it, or how it is implemented.

Now documentation, which can be written in four languages: English, Japanese, German, and Chinese, can be packaged into the X-Function. To do this, there are two X-Functions that can be used: **xf2doc**, and **doc2xf**. Note that these two X-Functions cannot be found from the menu, and can only be accessed from script. The X-Function **xf2doc** will unpack the help from the .oxf file and generate a Microsoft Word® document to the specified path (If the option **chm** is checked, either from the dialog or using script **xf2doc chm:=1**, a document with the same structure as the chm help will be generated. This is useful when you need to preview the help). Then the document can be edited with Microsoft Word®. After the documentation is finished, you need to save the document and then use **doc2xf** to package the document into the .oxf file.

When documentation is finished for an X-Function that meets the specifications and can run compatibly, smoothly, and robustly, an integrated X-Function is born. Now you can share the X-Function with others simply through the .oxf file.

19.3.4 Adding Custom Button on Dialog

19.3.4.1 Adding Custom Button on Dialog

The following example shows how to add two custom buttons: the Reset button and the Apply button, to an X-Function dialog, as well as an event1 function to catch what button is being clicked.

Press F10 to open X-Function Builder. Type "CustomButton" into the X-Function edit box as the X-Function name.

Click the  button to switch to the Tree View panel, and type "Reset|Apply" in the GetN Dialog Custom Buttons edit box. Click Save to save this X-Function.

Open the X-Function in Code Builder, and put the following code in the CustomButton_event1 function, then click the 

button.

```
if( nEventID == GETNE_ON_CUSTOM_BUTTON1 )
{
        out_str("Reset click");
}

if( nEventID == GETNE_ON_APPLY )
{
        out_str("Apply click");
}

if( nEventID == GETNE_ON_OK )
{
        out_str("OK click");
}
```

Run *CustomButton -d* in the Command window to open the dialog of this X-Function. There should be four buttons: Reset, Apply, OK and Cancel.

Click the Reset button, and "Reset click" gets printed in the Command window. If you click the Apply button, "Apply click" is printed and the Cancel button is changed to a Close button. Click OK, and "OK click" is printed and the dialog is closed.

19.3.4.2 Show Preview Button on Graph Preview GetN Dialog

X-Function framework supports a Show Preview button and an Auto Preview check box on the Graph Preview GetN dialog. These two controls are used to update the preview automatically (keep the Auto Preview check box checked) or manually (click the Preview button). The following steps demonstrate how to add the two controls.

The following five steps illustrate the process to add Preview button controls.

Press F10 to open X-Function Builder. Type "PreviewButtons" into the X-Function edit box as the X-Function name.

In the Variables table, set up a variable as below and then click the Save button to save this X-Function.

```
Name: gp

Label: Graph

Input/Output: Input

Data Type: string

Control: ...Graph
```

Run the command *PreviewButtons -d* to open this X-Function dialog. There are only two buttons: OK and Cancel. Click Cancel to close the dialog.

In X-Function Builder, click the  button to switch to the Tree View panel, set Usage Context -> Menus -> Auto GetN Dialog as GetNGraphBox with Preview, and check the Show Preview Button check box. Save this X-Function.

Run *PreviewButtons -d* to open the dialog again. Now there are two more controls displayed on the dialog: an Auto Preview check box and a Preview button.

The steps below show how to auto-update the dialog's preview graph according to the selection of the **gp** variable. For more details about the GetN Graph Preview Dialog, see X-Functions: Creating an X-Function: Creating Graph Preview GetN Dialog.

Open this X-Function in Code Builder, copy the following code, and paste it into the **GetNGraphPreview_OnChange** function.

```
//include the bit to call GetNGraphPreview_OnUpdateGraph to update graph

if( !(dwUpdateGraph & UPDATE_GRAPH_CHANGED) )

        dwUpdateGraph |= UPDATE_GRAPH_CHANGED;
```

Copy the codes below into the **GetNGraphPreview_OnUpdateGraph** function.

```
string strGraph = trGetN.gp.strVal;

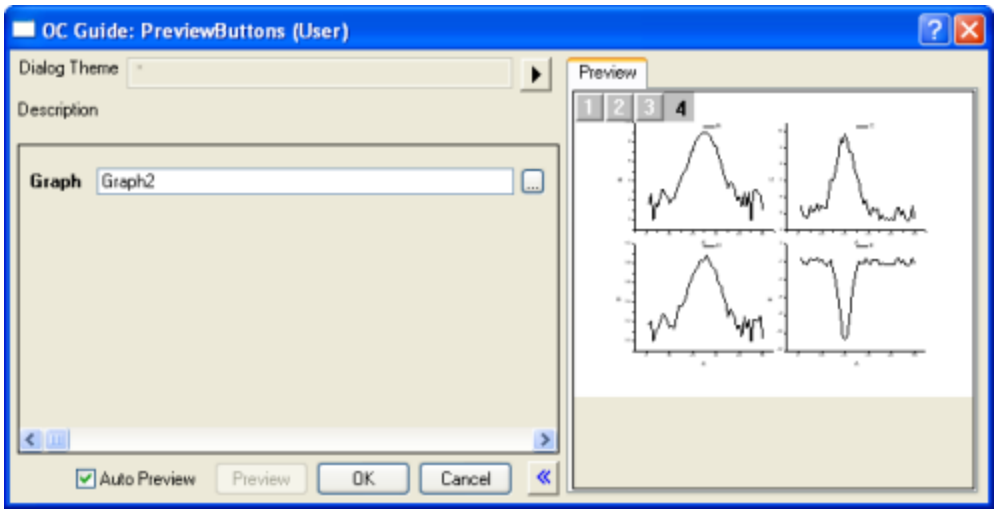
GraphPage gpSource(strGraph);

GraphPage gpDest(pgTemp);

page_clone(gpSource, gpDest);
```

Click the Compile button. Then run *PreviewButtons -d* in the Command window to open the dialog.

In the dialog, type the graph page name into the Graph control, or click the ... button to choose one graph page via the Graph Browser dialog. Click the Auto Preview check box or the Preview button, and the specified graph will display on the preview panel. See the picture below.



19.4 Using X-Functions

19.4.1 Using X-Functions

This section covers the following topics:

Calling X-Functions from Script

Calling X-Functions in Origin C

19.4.2 Calling X-Functions from Script

X-Functions can be called from LabTalk by default. This means that by default you can call an X-Function from the Command Window, Script Window, and anywhere else LabTalk script can be executed. This includes .OGS files, toolbar buttons, and more.

The LabTalk setting for an X-Function can be changed by doing the following:

Open the X-Function in X-Function Builder.

Activate Tree View by choosing it on the Tools menu or by clicking the Tree View toolbar button.

Expand the Usage Context branch to show the LabTalk setting.

When the check box is checked, it means the X-Function can be used from LabTalk.

19.4.2.1.1 Call an X-Function and show it's variables dialog

Passing the **-dialog** or **-d** option to an X-Function will open the X-Function's variables dialog. This dialog will allow you to set variables using the dialog's interface.

The following will open the smooth X-Function variables dialog using all the default values.

```
smooth -d;
```

You can set some variables in the command line and still open the variables dialog. The variables that were not set in the command line will use their default values.

```
smooth (1,2) method:=2 -d;
```

19.4.2.1.2 Call an X-Function and do not show it's variables dialog

The variables dialog is not necessary to set variables. All variables can be set in the command line. Again, any variables that are not set in the command line will use their default values.

The following will call the smooth X-Function to smooth the data in columns 1 and 2 using the "Savitzky-Golay" method, with a 5-point window and the boundary condition set to "reflect".

```
smooth (1,2) npts:=5 method:=SG b:=1;
```

The variable assignment can be done in multiple ways:

By variable index. Assign values without a variable name. For example, (1,2) assigns the data in the active Worksheet columns 1 and 2 to the **iy** variable. **iy** is the first variable.

Use the default value. No assignment is necessary to use the default value. Here if you do not assign a value to the **weight** variable, the default value is 0.

By the variable's full name. For example, npts:=5, assigns a numeric value of 5 to the **npts** variable; method:=SG, assigns enumeration to the **method** variable.

By the first letter of the variable. For example, b:=1. This usage is very simple and can avoid having to remember the long variable names, but you need to make sure the first letter of each of the variables is unique.

19.4.2.1.3 Call an X-Function and set variables with a theme

The variable settings for an X-Function can be saved into a theme. You can call an X-Function and have it use a saved theme by using the **-theme** or **-t** option.

```
smooth (1,2) -t mytheme; // Call smooth and use the 'mytheme' theme

smooth -t mytheme -d; // Open dialog and load the 'mytheme' theme
```

For more details please see the **X-Functions: Accessing X-Functions from Script** chapter in Origin's Help.

19.4.3 Calling X-Functions in Origin C

Origin comes with many X-Functions for handling a variety of tasks. X-Functions can be called from both LabTalk and Origin C. This Section will show you how to call X-Functions from Origin C using Origin C's XFBase class. This mechanism also can be used in calling one X-Function in another X-Function.

The XFBase class is declared in the XFBase.h header file located in the Origin C System folder. The XFBase.h header file is not included in the Origin.h header file and must be included separately in any Origin C file that uses the XFBase class.

```
#include <XFBase.h>
```

19.4.3.1 Calling the impFile X-Function From Origin C

The following Origin C code defines a general function for importing files into Origin. The function takes two arguments: a data file name and an import filter file name. The function first creates an instance of the XFBase class, constructed with the name of the X-Function to be called. In this case, the X-Function name is impFile. It then sets the X-Function arguments using the *SetArg* method, and finally calls the X-Function using the *Evaluate* method.

```
bool call_impFile_XF(LPCSTR lpcszDataFile, LPCSTR lpcszFilterFile)
{
    string strDataFile = lpcszDataFile;
    string strFilterFile = lpcszFilterFile;

    // Create an instance of XFBase using the X-Function name.
    XFBase xf("impFile");
```

```
    if (!xf)
        return false;

    // Set the 'fname' argument.
    if (!xf.SetArg("fname", strDataFile))
        return false;

    // Set the 'filtername' argument.
    if (!xf.SetArg("filtername", strFilterFile))
        return false;

    // Call XFBase's 'Evaluate' method to execute the X-Function
    if (!xf.Evaluate())
        return false;

    return true;
}
```

The following Origin C code shows how to call our call_impFile_XF function defined above. We will call it to import an image file.

```
// Lets import the Car bitmap located in Origin's Samples folder.
string strImageFile = GetAppPath(TRUE) +
    "Samples\\Image Processing and Analysis\\Car.bmp";

// Lets import the bitmap using the Image import filter.
string strFilterFile = GetAppPath(TRUE) + "Filters\\Image.oif";
```

```
// Let's call our general function that will call the impFile X-Function.

call_impFile_XF(strImageFile, strFilterFile);
```

19.5 X-Function Examples

19.5.1 X-Function Examples

This section covers the following topics:

Construct Controls on X-Function GUI

Open Graph Browser Dialog in X-Function

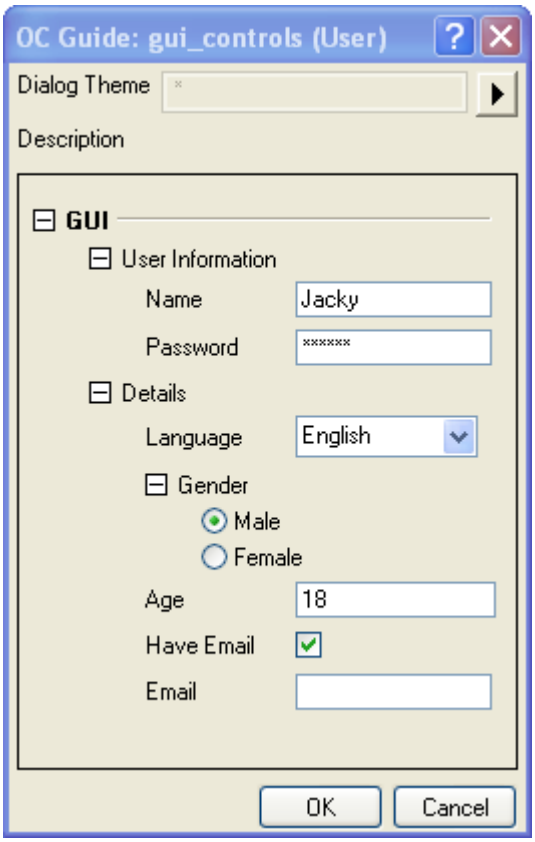
X-Function ReportTree Example

X-Function ReportData Example

19.5.2 Construct Controls on X-Function GUI

19.5.2.1 Summary

This example demonstrates how to construct controls in an X-Function GetN dialog by TreeNode variable. This mechanism is used to avoid adding controls by multiple variables. This is an example X-Function dialog:



19.5.2.2 You will learn

How to create different controls by make_tree.

How to update a GUI on a specified control event.

How to check and ask for valid input.

19.5.2.3 Steps

Select **Tools: X-Function Builder** to open the X-Function Builder dialog, select **File: New X-Function Wizard** to create an X-Function with only one Input Variable named **GUI** of TreeNode type, and save it as gui_controls.oxf.

Open the X-Function in Code Builder, and add the following code into the *gui_controls_make_tree* function.

```
if ( strcmp("GUI", lpcszVarName) == 0 ) // check variable by name
{
        // ID should be unique
```

```
        int nBranchID = 0x0001;

        int nUserID = GET_USER_DATAID(0); //NodeID to enable Theme support



        GETN_USE(tr)  //use this node as current node



        //begin of sub branch

        GETN_BEGIN_BRANCH(UserInfo, "User Information")
GETN_ID(nBranchID++)

                // string edit box

                GETN_STR(UserName, "Name", "Unknown") GETN_ID(nUserID++)



                // pass word edit box

                GETN_PASSWORD(Password, "Password", "") GETN_ID(nUserID++)

        GETN_END_BRANCH(UserInfo) //end of sub branch



        //begin of sub branch

        GETN_BEGIN_BRANCH(Detail, "Details") GETN_ID(nBranchID++)

                // editable drop-down list

                GETN_STRLIST(Language, "Language", "English",
"|English|German")

                GETN_ID(nUserID++)



                // radio buttons

                GETN_BEGIN_BRANCH(lGender, "Gender") GETN_ID(nBranchID++)

                        GETN_RADIO_INDEX(Gender, 0, "Male|Female")
GETN_ID(nUserID++)

                        // the format of radio buttons

                        GETN_OPTION_DISPLAY_FORMAT(DISPLAY_EDITOR_LEFT)
```

```
                GETN_END_BRANCH(lGender)



                //numeric edit box, age must be Integer, so use format "%d"

                GETN_NUM(Age, "Age", 18) GETN_OPTION_NUM_FORMAT("%d")

                GETN_ID(nUserID++)



                //check box with event handling function

                GETN_CHECK(iEmail, "Have Email", 0)
GETN_ID(nUserID++)//check box

                        GETN_OPTION_EVENT(_show_email) //event of iEmail
checkbox



                // string edit box

                GETN_STR(Email, "Email", "") GETN_ID(nUserID++)

        GETN_END_BRANCH(Detail) //end of sub branch

}
```

Add the following event handling functions after the line *//put your own support static functions here*.

```

static bool _show_email(TreeNode& tr, int nRow, int nType, Dialog& Dlg)

{

        TreeNode trDetail = tr.GUI.Detail;

        if ( !trDetail )

                return false;

        int nShow = trDetail.iEmail.nVal;

        //if check, show, else hide.

        trDetail.Email.Show = (nShow == 1);

        return true;

}
```

```
static bool _check_email(LPCSTR lpcszEmail)
{
        string strEmail(lpcszEmail);
        if ( strEmail.Count('@') != 1 )
                return false; //should contains '@' and only one.
        int nSep = strEmail.Find('@');
        string strLeft = strEmail.Left(nSep);
        string strRight = strEmail.Right(strEmail.GetLength() - nSep - 1);
        if ( strLeft.GetLength() == 0 )
                return false;
        if ( strRight.GetLength() == 0 )
                return false;
        LPCSTR lpcszInvalid = "~!#$%^&*()+ -=|\\/><,`";
        if ( strEmail.FindOneOf(lpcszInvalid) >= 0 )
                return false;
        return true;
}
```

Add error checking code into the *gui_controls_event1* function.

```
if ( strcmp(lpcszNodeName, "Email") == 0 )
{
    string strVal = trGetN.GUI.Detail.Email.strVal;
    if ( !_check_email(strVal) )
    {
        strErrMsg = "Invalid Email Address! Please correct it";
```

```
        //disable OK button, should not continue when there is error.
        bOKEnable = false;
    }
}
```

Change the return value from *return false* (default) to *return true* to force a GUI change.

In the X-Function body, we just print out the user settings by tree. Click the Compile button 

to compile and save changes.

```
out_tree(GUI);
```

19.5.2.4 Run the X-Function

Run the *gui_controls -d* command in the Script Window.

Click the Have Email check box on the dialog, and an email edit box will display.

If you input an invalid email address to the email edit box, for example, type in *abc*, an error message will show on the bottom of the dialog and the OK button will be disabled.

Click OK. The setting tree will be outputted to the Script Window.

19.5.3 Open Graph Browser Dialog in X-Function

19.5.3.1 Summary

This example demonstrates how to add a button to open the Graph Browser dialog to pick a graph in the X-Function GetN dialog, and then how to customize the Graph Browser dialog by filter function.

19.5.3.2 You will learn

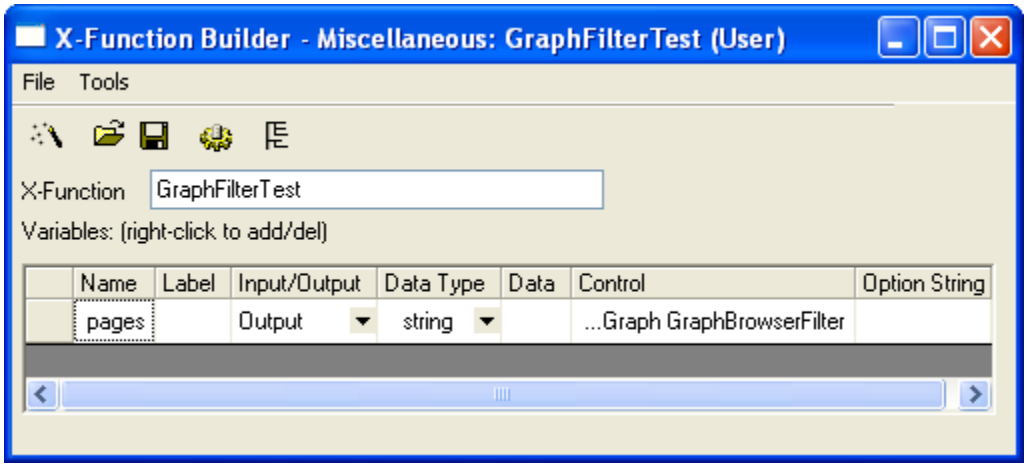
How to add a button to open a Graph Browser in the X-Function dialog.

How to customize the Graph Browser dialog by filter function.

19.5.3.3 Steps

The following three steps will show how to add a button in the X-Function dialog to open a Graph Browser dialog without the use of a filter function. This is the simplest process, and all settings in the Graph Browser dialog will be set to their defaults.

Select **Tools: X-Function Builder** from the Origin menu. Create an X-Function **GraphFilterTest** as follows.



Note the control string is *...Graph GraphBrowserFilter*.

Click Edit X-Function in the Code Builder toolbar, and type *out_str(pages);* in the main function, GraphFilterTest. Click the Compile button and then click Return to Dialog.

Click the Save OXF File toolbar button to save. Run *GraphFilterTest -d* in the Command window, and when a dialog shows, click the ... button to choose a graph, then click OK to close the dialog. The name of the selected graph window will be printed in the Command window.

If you do NOT want to customize the Graph Browser Dialog, then there is no need to go on; the following steps are all about customizing the Graph Browser dialog, for example, setting the default view mode to Tree view. Create a GraphFilter.c file and save it in *<Origin installation folder>\OriginC\OriginLab*. The source file should include the following code.

```
#include <..\Originlab\GraphFilter.h> // this is an Origin built-in file

int GraphBrowserFilter(int nMsg, Page& pg = NULL)

{

        switch(nMsg)

        {

        case GBFM_IS_USE_PAGE: // return 1 for true.

                if(!pg)

                        return -1;

                // filter page including multiple layers
```

```
                return pg.Layers.Count() > 1? 0 : 1;


        case GBFM_SHOW_EMBED_PAGE_CHKBOX:
                // see other options in GraphFilter.h
                return EMBED_CHECKBOX_ENABLE;


        case GBFM_SHOW_SWITCH_MODE:
                // see other options in GraphFilter.h
                return VIEW_MODE_BOTH_WITH_TREE_DEFAULT;


        case GBFM_SORT_PAGES:
                return 1; // return 0 or 1
        }
        return -1; // error
}
```

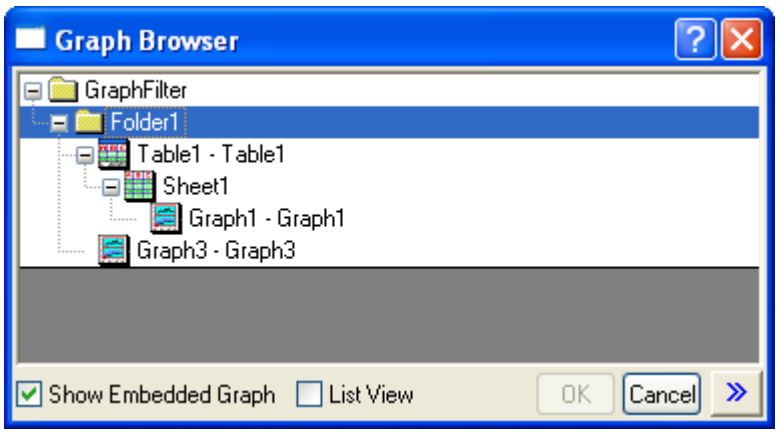
Create a header file named MyGraphFilter.h and save it in *<Origin installation folder>\OriginC\OriginLab*.

Open this X-Function in X-Function Builder, and click the *Edit X-Function in Code Builder* button, to include the newly created header file, from above. Note that MyGraphFilter.c and MyGraphFilter.h can also be put in another folder, but you have to make sure that they are in the same folder as each other. If they are in another folder, an absolute path should be used to include the header file, e.g.

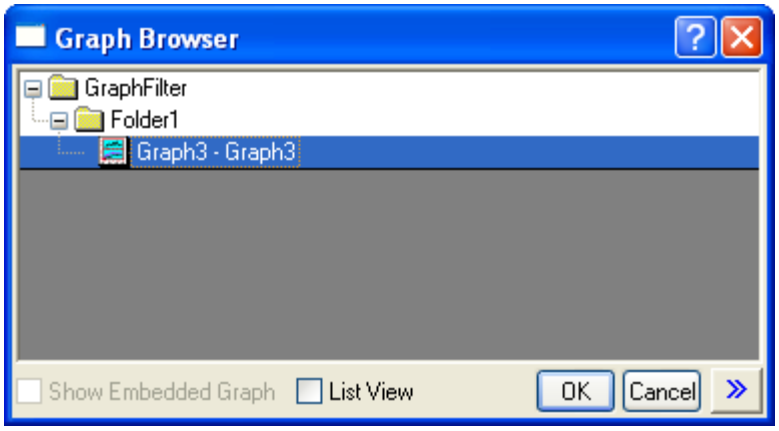
```
//Case 1. if header file is located in the OriginC\Originlab\ folder
//put additional include files here
#include <..\Originlab\MyGraphFilter.h>


//Case 2. if header file is NOT located in the OriginC\Originlab\ folder
//put additional include files here
#include "D:\MyFolder\MyGraphFilter.h"
```

Run *GraphFilterTest -d* in the Command Window, and click the Browse button in the GraphFilterTest dialog. You can see that the *Show Embedded Graph* check box is enabled and selected, and the embedded graph is listed in the tree browser.



Open MyGraphFilter.c from <*Origin installation folder*>*OriginC**Originlab*\ with the Code Builder. Change the return value from *EMBED_CHECKBOX_ENABLE* to *EMBED_CHECKBOX_DISABLE_EMBEDDING_PAGE_NEVER_ENABLE* for the GBFM_SHOW_EMBED_PAGE_CHKBOX case. Click the Compile button to compile MyGraphFilter.c. Then run *GraphFilterTest -d* in the Command Window again. The *Show Embedded Graph* check box is disabled and unchecked now, and the embedded graph is not listed in the tree browser.



You can try to use other enum values in <*Origin installation folder*>*OriginC**Originlab**GraphFilter.h* as the return value for other cases in MyGraphFilter.c. Please remember to compile MyGraphFilter.c after each change.

The built-in X-Functions containing Graph Browsers (e.g. Import and Export X-Function **expGraph**) can be customized using the same method.

19.5.4 X-Function ReportTree Example

19.5.4.1 Summary

In this example, we will show how to do statistics on the selected data in a worksheet, and generate a report in a new hierarchy sheet. The selected data can be one column, multiple columns, or any one subrange in the worksheet.

As the output data type is ReportTree, the dialog will have a Recalculate combo box, which can set the recalculate mode of the report sheet to "Manual", "Auto Update" or "None".

19.5.4.2 You will learn

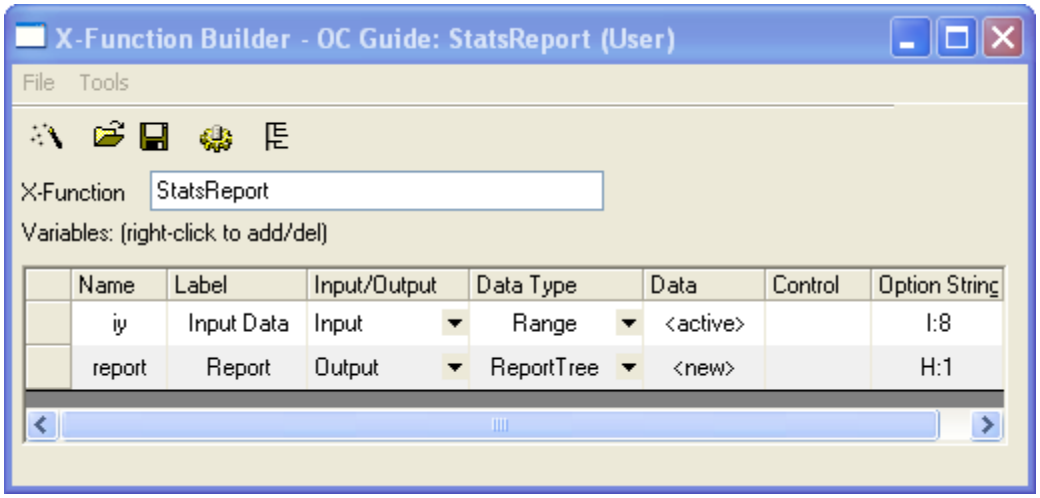
How to get data from the selected data range.

How to do error handling in event1, before_execute and the main function.

How to generate report tables in a hierarchy sheet with a recalculation lock.

19.5.4.3 Steps

Hit F10 to open X-Function Builder, enter the X-Function name and variables, as in the following picture, and then click Save. .



Open this X-Function in Code Builder to edit the source code. First include the needed header file, as below.

```
#include <ReportTree.h> // needed for ReportTable class
```

Add error checking code into *StatsReport_event1* to check the input data range, like in the following.

```
DataRange drInput;

drInput.Create(trGetN.iy.strVal);
```

```
// if input is invalid, show error message
// on the bottom of dialog
// and disable OK button
if ( !drInput.IsValid() || drInput.GetNumRanges() < 1 )
{
        strErrMsg = "Please select valid data for input";
        bOKEnable = false;
}
```

Add error checking code in *report_stats_before_execute*.

```
DataRange drInput;
drInput.Create(trGetN.iy.strVal);

// if input is invalid, print out error message
// and abort the X-Function execution
if ( !drInput.IsValid() || drInput.GetNumRanges() < 1 )
{
        out_str("Invalid Input Data");
        nRet = XFEVT_ABORT;
}
```

Add a static function to check the input range after the line *//put your own support static functions here*

```
static      bool       _check_input(const Range& iy)
{
        int nRanges;

        if ( !iy.IsValid() )
```

```
                return false;


        nRanges = iy.GetNumData(DRR_COLUMN_INDEX |
                                DRR_NO_FACTORS);


        if ( nRanges <= 0 )
                return false;
        return true;
}
```

Add any needed macros under the static function. The macros will be used in the X-Function main function.

```

// ID can be any value, but must be unique.
#define    TABLE_ID_BEGIN          0x1000
#define    ROW_ID_BEGIN            0x0001
```

In the X-Function main function *StatsReport*, add the following code to get the data from the specified data range, do statistics, and generate a report sheet.

```

if ( !_check_input(iy) )
{
        // if input is not valid,
        // show error message and
        // abort X-Function execution.
        XF_THROW("Invalid input data");
        return;
}


//create table to show statistics summary.
```

```
ReportTable rt = report.CreateTable("Summary",_L("Summary"),
                                    TABLE_ID_BEGIN);


//report table's column headers.
const vector<string> vsColLabels = {
        "N",
        "Number of Missing",
        "Mean",
        "SD",
        "SEM",
        "Sum",
        "Variance"
};
int nRowID = ROW_ID_BEGIN;
int nRanges = iy.GetNumData(DRR_COLUMN_INDEX |
                            DRR_NO_FACTORS);


for ( int nRange = 0; nRange < nRanges; nRange++ )
{
        // get the subrange - one column
        DataRange drOne;
        iy.GetSubRange(drOne, DRR_COLUMN_INDEX |
                        DRR_NO_FACTORS, nRange);


        // get range string, like [Book1]Sheet1!A
        string strDataLabel;
```

```
        drOne.GetRangeString(strDataLabel);


        vector vInput;
        drOne.GetData(&vInput, 0);
        if ( vInput.GetSize() == 0 )
        {
                // print out warning message when column is empty
                // and then go to the next column
                warning_msg_box(
                   strDataLabel + ", empty column found.",
                   false, 'W');
                continue;
        }


        int N, Missing;
        double dMean, dSum, dVariance, dSD, dSE;
        int nRet = ocmath_basic_summary_stats(vInput.GetSize(),
                        vInput,
                        &N, &dMean, &dSD, &dSE, &dVariance, &dSum,
                        NULL, NULL, NULL, NULL, NULL, NULL, &Missing);
        if ( STATS_NO_ERROR != nRet )
        {
                // print out warning message when
                // statistics function failed.
                warning_msg_box(
                   strDataLabel + ", statistics fails.",
```

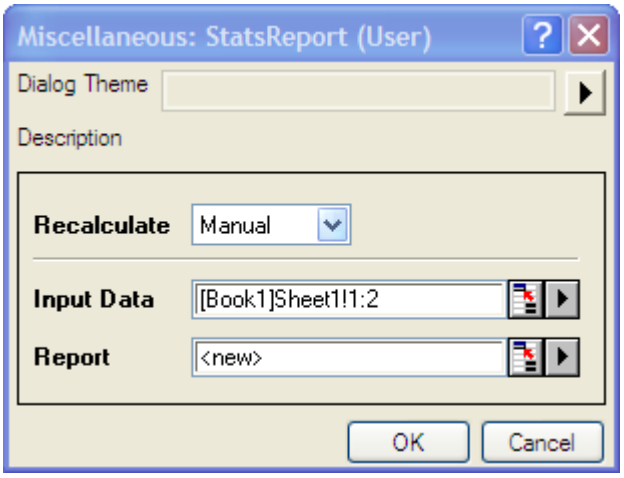
```
                    false, 'W');
                continue;
        }

        vector vResults(vsColLabels.GetSize());
        vResults[0] = N;
        vResults[1] = Missing;
        vResults[2] = dMean;
        vResults[3] = dSD;
        vResults[4] = dSE;
        vResults[5] = dSum;
        vResults[6] = dVariance;

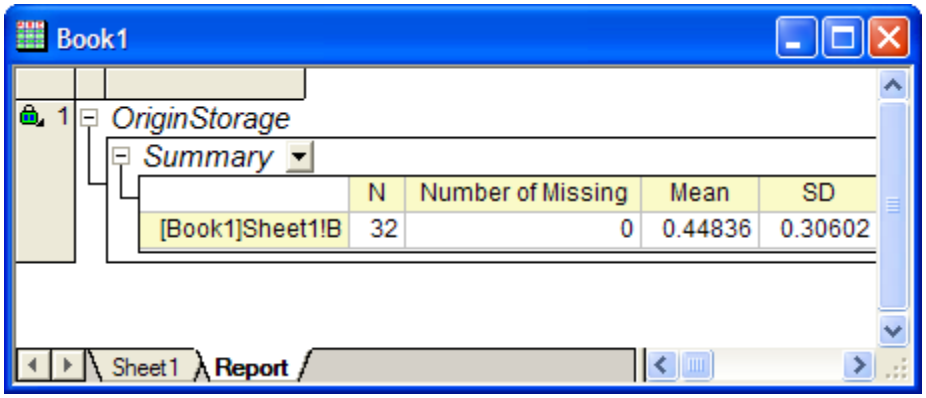
        //add new row to report table.
        string strName = "Data" + nRange;
        rt.AddRow(strName, vResults, strDataLabel,
                vsColLabels, NULL, nRowID++);
}
```

19.5.4.4 Run the X-Function

Keep one Worksheet active with some data, highlight two columns, and type *StatsReport -d* in the Script Window to open the X-Function dialog.



Click OK , and a new report sheet will be generated.



19.5.5 X-Function ReportData Example

19.5.5.1 Summary

This example demonstrates how to get multiple XY data sets, do linear fitting on each XY, and put the fitted data into a new result sheet.

As the output data type is ReportTree, the dialog will have a Recalculate combo box, which can set the recalculate mode of the report sheet to "Manual", "Auto Update" or "None".

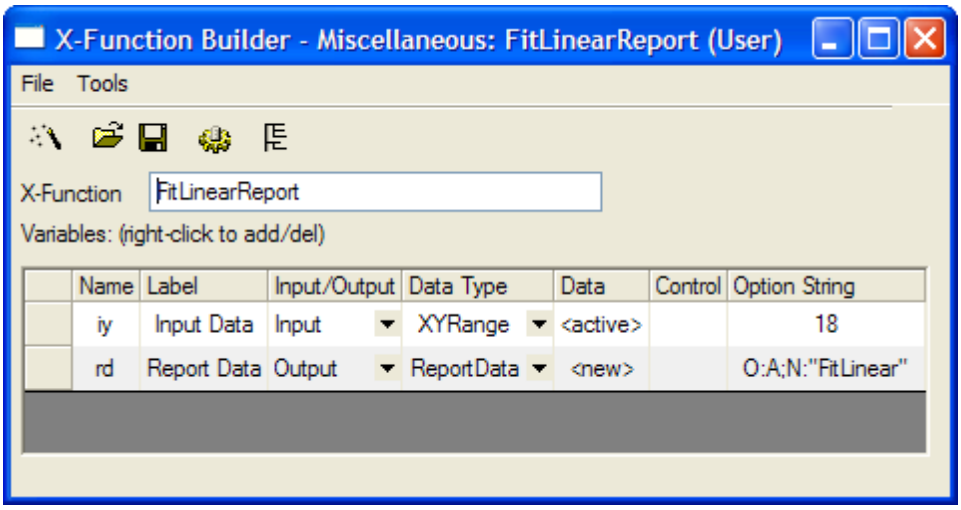
19.5.5.2 You will learn

How to work on multiple XY data sets.

How to report data in a new result sheet with a recalculation lock.

19.5.5.3 Steps

Hit F10 to open the X-Function Builder, enter the X-Function name and variables, as in the following picture, and then click Save. .



Open this X-Function in Code Builder to edit the source code. First include the needed header file, as below.

```
#include <ReportTree.h> // needed for ReportTable class
```

Add any needed macros under the line *//put your own support static functions here*. The macros will be used in the X-Function's main function.

```
// ID can be any value, but must be unique.
#define TABLE_ID                    1
#define SUBNODE_ID_BEGIN    1000
```

In the X-Function's main function, *FitLinearReport*, add the following code to get the XY data from the specified data range, do linear fitting on each XY, and generate a report sheet.

```
// create report table
ReportTable     rt;
rt = rd.CreateTable("ReportData", "Fitted Data", TABLE_ID);
int nSubID = SUBNODE_ID_BEGIN;
```

```
DWORD dwRules = DRR_GET_DEPENDENT | DRR_NO_FACTORS;
int nNumData = iy.GetNumData(dwRules);


for(int nRange = 0; nRange < nNumData; nRange++)
{
        DataRange drOne;
        iy.GetSubRange(drOne, dwRules, nRange);


        vector vx, vy;
        drOne.GetData(dwRules, 0, NULL, NULL, &vy, &vx);


        // there are two parameters in linear fitting
        FitParameter sFitParameter[2];
        if( STATS_NO_ERROR == ocmath_linear_fit(vx, vy,
                                vy.GetSize(), sFitParameter) )
        {
                // add fitted X data to report table
                string strName = "X" + (nRange+1);
                string strLongName = "X";
                rt.AddColumn(vx, strName, nSubID++, strLongName,
                                OKDATAOBJ_DESIGNATION_X);


                // calculate fitted Y data
                double dIntercept = sFitParameter[0].Value;
                double dSlope = sFitParameter[1].Value;
                vector vFitY;
```

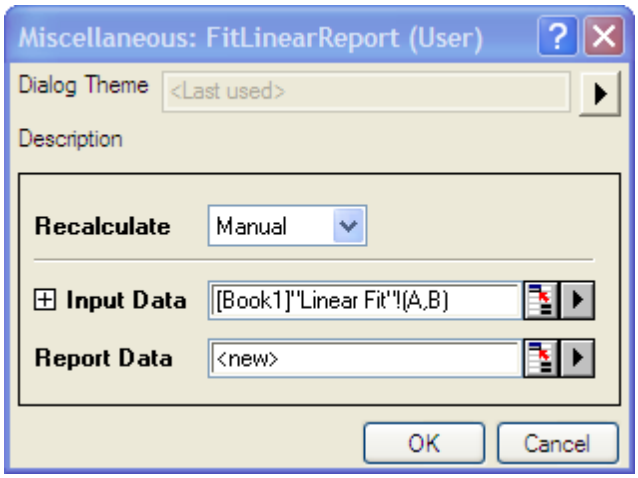
```
                vFitY = vx * dSlope + dIntercept;


                // add fitted Y data to report table
                strName = "Y" + (nRange+1);
                string strRange;
                strRange = drOne.GetDescription(GETLC_COL_LN_ONLY);
                strLongName = "Fitted data on " + strRange;
                rt.AddColumn(vFitY, strName, nSubID++, strLongName,
                                OKDATAOBJ_DESIGNATION_Y);
        }
}
```

19.5.5.4 Run the X-Function

Import Linear Fit.dat from OriginExePath\Samples\Curve Fitting to a Worksheet.

Highlight all columns and run *FitLinearReport -d* in the Script window, and then you can see the dialog below.



Accept the default settings, and click the OK button. A report sheet will be generated with three group-fitting XY data sets.

LinearFit - Linear Fit.dat

	X1(X)	Y1(Y)
Comments	Fitted Data	Fitted Data
Long Name	X	Fitted data on B
1	1	1.01566
2	1.5	1.64507
3	2	2.27449
4	2.5	2.9039
5	3	3.53332
6	3.5	4.16273
7	4	4.79215
8	4.5	5.42156
9	5	6.05098
10	5.5	6.6804
11	6	7.30981
12	6.5	7.93923
13	7	8.56864
14	7.5	9.19806
15	8	9.82747

Linear Fit | FitLinear

20 Accessing External Resources

20.1 Accessing External Resources

Origin C can access external DLLs and, in addition, applications outside of Origin can be added using automation (COM) server capability.

This section covers the following topics:

Calling Third Party DLL Functions

Access an External Application

20.2 Calling Third Party DLL Functions

20.2.1 Calling Third Party DLL Functions

20.2.1.1 Declaration

Origin C can make calls to functions **(C linkage only)** 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
```

20.2.1.2 Loading DLL

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 Origin exe folder

#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.

20.2.1.3 Version Control

To make sure the external dll works correctly, the 32-bit dll is for 32-bit version of Origin, and the same for the 64-bit version. **#ifdef _OWIN64** is used to detect which version (32-bit or 64-bit) of current Origin is, so to determine which version of dll to be loaded. For example,

```
#ifdef      _OWIN64

#pragma dll(UserFunc_64, header)

#else

#pragma dll(UserFunc, header)

#endif     //_OWIN64
```

20.2.1.4 Examples

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. These examples can be found in this zip file, under the *\Programming Guide\Calling Fortran* and *\Programming Guide\Calling C DLL* subfolders.

This section covers the following topics:

Calling GNU Scientific Library

Access CPlusPlus(.Net) And CSharp DLL

Access Python via External DLL

20.2.2 Calling GNU Scientific Library

This article demonstrate how to use GSL in Origin C.

> **Note:** To use Visual Studio build 64bit dlls from a GSL source code of v2.5:
>
> Get files from https://github.com/BrianGladman/gsl
>
> Follow https://github.com/BrianGladman/gsl/blob/master/build.vc/gsl.readme.txt instruction to build the project/solution to get gsl.dll and cblas.dll.
>
> gsl.dll is libgsl.dll mentioned below

First you need download the GSL 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\. When using the downloaded dlls, please pay attention to the version issues.

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.

20.2.2.1.1 ocgsl.h

```
// when loading the dll, need to load the correct version,
// see the "version issues" link above for more details
#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);
```

Note: Please don't write **#pragma dll(libgslcblas, header)** in code, or it may cause compile error.

20.2.2.1.2 test_gsl.c

The following is a simple OC file to show how to call gsl_sf_zeta_int and gsl_fit_linear

```
#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);

}
```

20.2.2.2 Using GSL in a Fitting Function

There is also an exampleto show how to use gsl functions in a fitting function.

20.2.2.2.1 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;
```

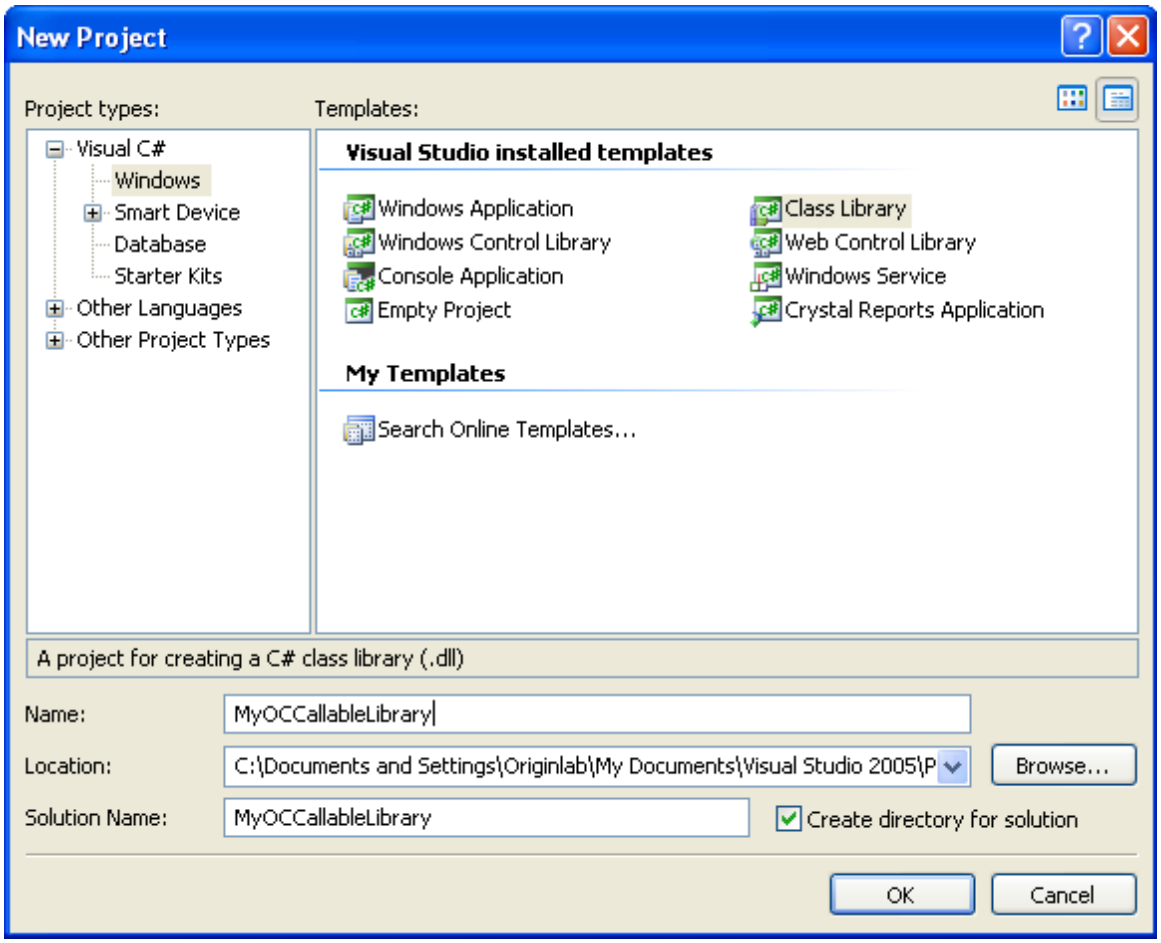
20.2.3 Access CPlusPlus(.Net) And CSharp DLL

This chapter will introduce how to access the DLL created by C++(.Net) or C# in Origin C.

20.2.3.1 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 provides a class with properties and functions. Function Sum shows how to pass data array from Origin C vector to C# function.

Run Microsoft Visual Studio as administrator, select File -> New -> Project..., in New Project dialog, choose the settings as below:



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;
```

```
        }

    }

}
```

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.

Right click on the project and select Properties to open the property page. Go to Application tab, and click Assembly Information button, then check the check-box of Make assembly COM-Visible. If for 32 bit version, go to the Build tab, and select the check-box of Register for COM interop. If for 64 bit version, go to Build Events tab, and then copy and paste the following command line to the Post-build event command line text box. Press F6 to build solution.

```
"%Windir%\Microsoft.NET\Framework64\v4.0.30319\regasm" "$(TargetPath)"
/CodeBase
```

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));

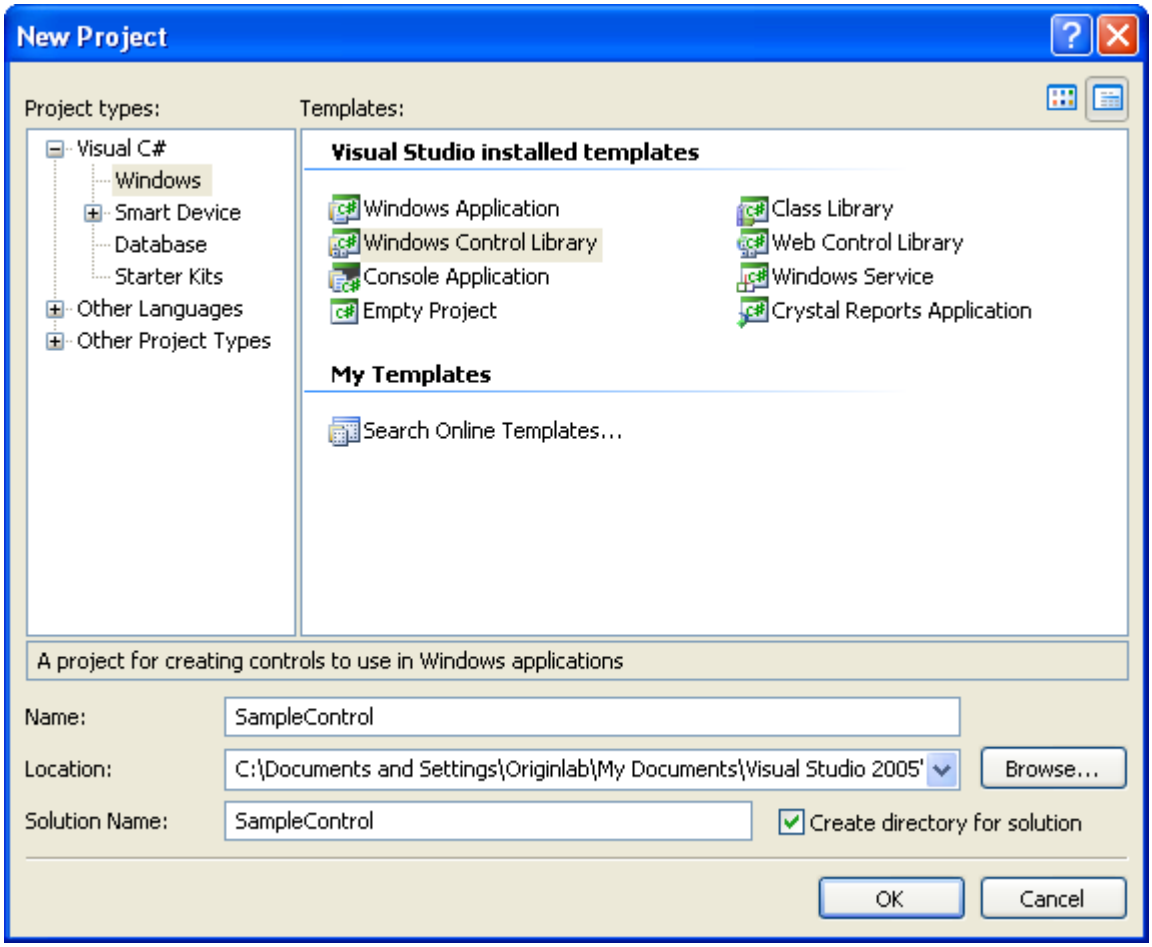
}
```

20.2.3.2 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.

Start Microsoft Visual Studio 2005, choose File -> New -> Project... to open New Project dialog, do the following settings, then click OK button.



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);
    }
}
```

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("...")].

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.

In UserControl1I.cs file, copy the following code outside UserControl1 class.

```
//declare delegates for events
public delegate void MouseAction(int x, int y);
```

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);
      }
}
```

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);
}
```

Create a resource-only DLL in Visual Studio 2008 as explained in Dialog Builder: Simple Hello World Dialog: Create Resource-only DLL in Visual Studio 2008

Use an ActiveX control in a dialog and show dialog by Origin C. In Origin Code Builder, create 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;
};
```

```
// Do not specify DLL file path here. 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();
}
```

20.2.4 Access Python via External DLL

In Origin C, there is no way to call Python functions directly. However, if you want to reuse your Python code in Origin C, it is recommended to wrap the Python functions in a DLL first, and then expose the functions in the DLL to Origin C code. In the following sections of this chapter, an example will be shown on how to do this step by step.

Notes: From Origin 2015, we can run python in Origin (support both command line and .py file), and use a **PyOrigin** module to access Origin from Python. view Python.chm for more details.

20.2.4.1 Running Environment

The example shown below is based on the following environment.

Windows 7

Python 3.3.5 (Assume the Python settings are already OK.)

Visual Studio 2012

20.2.4.2 Python File

The Python functions defined in the following are going to be reused in Origin C.

```
def SayHello():
    print("Welcome to use Python in your Origin C programs!") #actually this string
will not show in Origin C
    return 12345
```

```
def Add(a, b):
    return a + b


def Sub(a, b):
    return a - b


def Mult(a, b):
    return a * b


def Div(a, b):
    return a / b


def Mod(a, b):
    return a % b
```

First of all, go to the Python installed directory, then create a Python file under this directory, say "myLib.py". Then open this newly created file using a text editor, Notepad for example, and put the Python code above to this file, and save.

Start Windows Console (cmd.exe), and then switch the current path to the directory where Python installed.

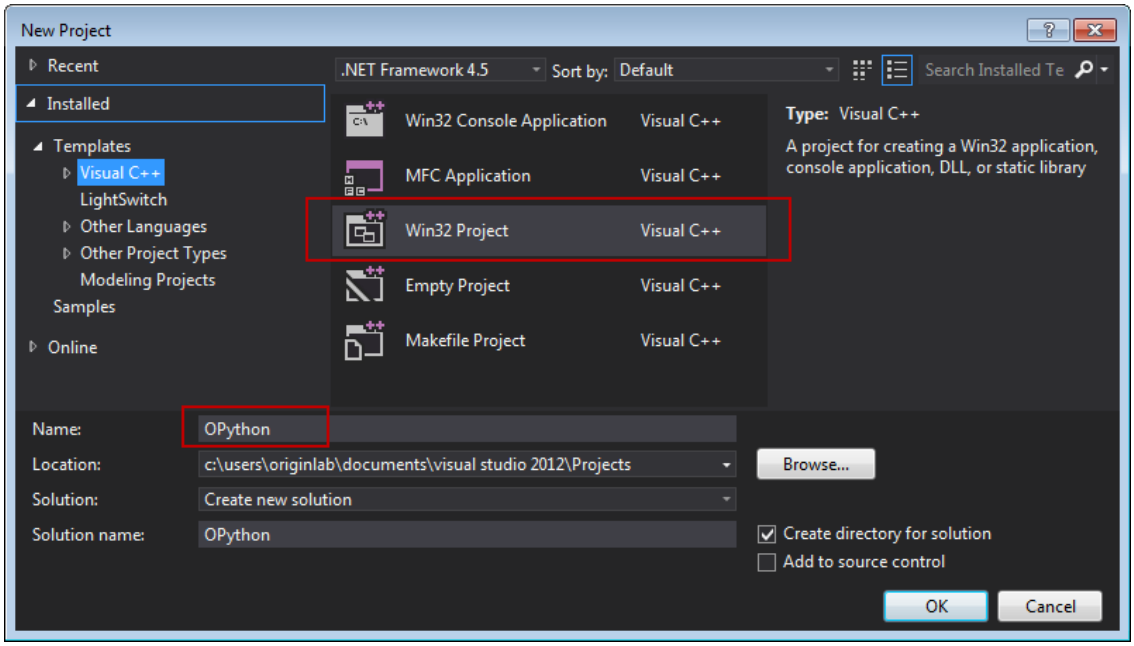
Run the following command to compile the Python file created just now.

```
python -m py_compile myLib.py
```

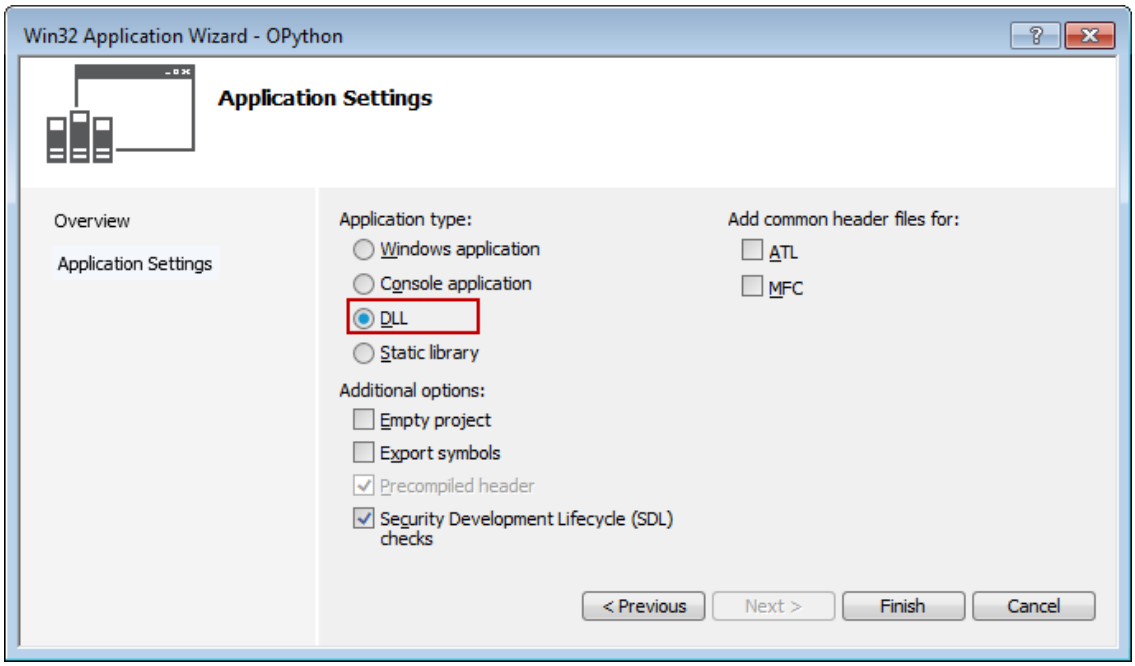
If successfully, there will be a pyc file (myLib.cpython-33.pyc, or something similar with "myLib") under this folder <Python Installation Directory>/__pycache__/ by default.

20.2.4.3 Build DLL

Start Visual Studio 2012, and create a new Win32 Project, named OPython.



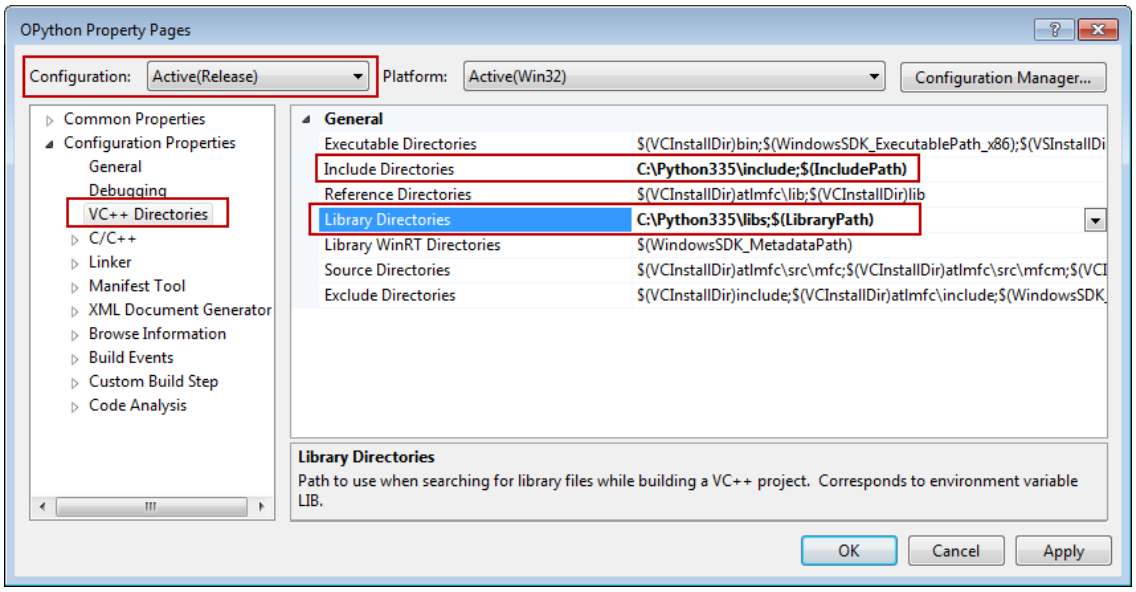
Click OK, and select the Application type to be DLL, and click Finish to create the project.



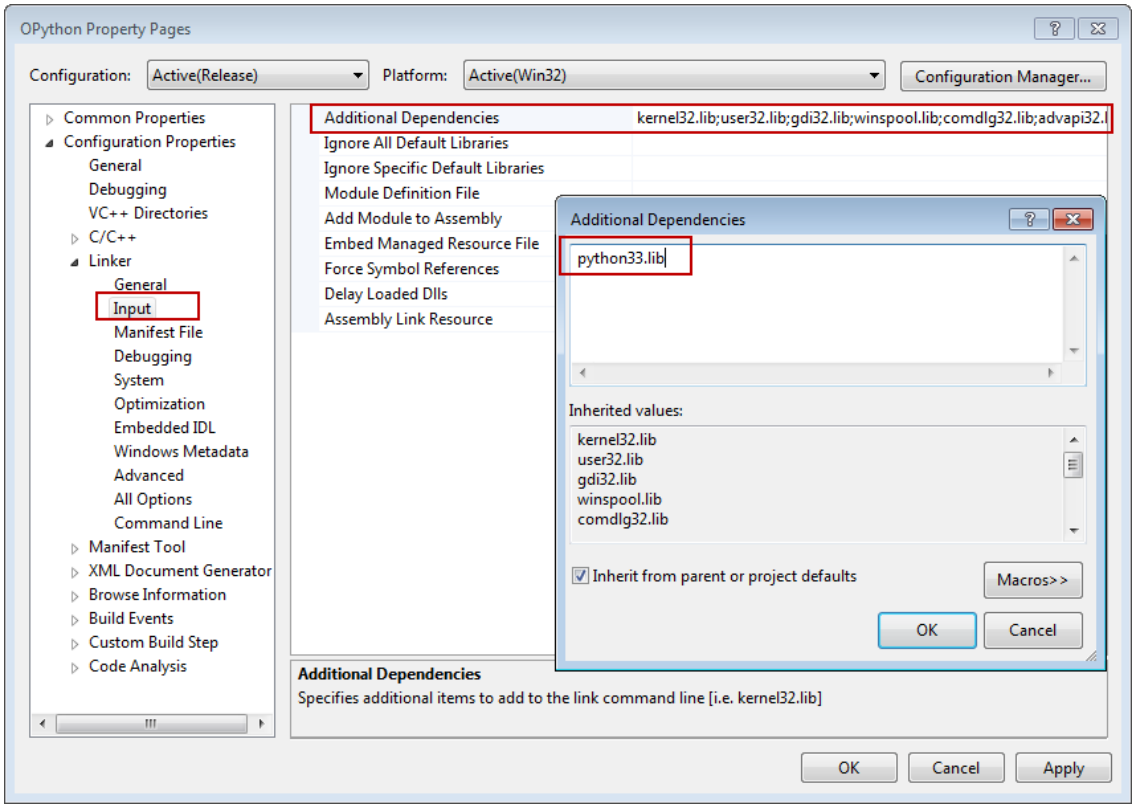
Set the Solution Configurations to be **Release**. Right click on the project, and choose **Property** from the context menu, to open the Property dialog.

Note: Here the 32-bit DLL will be created, and that will be used in 32-bit version of Origin.

In the Property dialog, from the left panel, activate **Configuration Properties: VC++ Directories**, and then in the right panel, add the Python header file path and library path to **Include Directories** and **Library Directories** respectively.



From the left panel, activate **Configuration Properties: Linker: Input**, and then in the right panel, add the Python library to **Additional Dependencies**, here is **python33.lib**. Apply all the settings and click OK.



Add a header file, named OPython.h, to the project, and put the following code to it.

```
#ifndef  _OPYTHON_H_

#define    _OPYTHON_H_



#ifdef     _OPYTHON_CPP_

        #define    OP_API  __declspec(dllexport) //use in VC

#else

        #define    OP_API
//use in OC

        #pragma dll(OPython) //this line is important, when use in OC,
Origin will find function body by searching in this DLL. Note:OPythonDLL,
is the generated dll, without extension name.

#endif



OP_API  int          PY_SayHello();



OP_API  float PY_Add(float a, float b);



OP_API  float PY_Sub(float a, float b);



OP_API  float PY_Mult(float a, float b);



OP_API  float PY_Div(float a, float b);



OP_API  float PY_Mod(float a, float b);



#endif     //_OPYTHON_H_
```

Open the OPython.cpp, which is created automatically when creating the project (If no such file, create it). Replace the original code by the following code.

```
#include "stdafx.h"

#define    _OPYTHON_CPP_ //use this macro to identify whether OPython.h is
included in VC(when create the DLL) or OC(when use the DLL)

#include "OPython.h"

#include <Python.h>



class PythonManager

{

public:

        PythonManager(); //init python environment

        ~PythonManager(); //clean python environment

};



PythonManager::PythonManager()

{

        Py_Initialize();

}



PythonManager::~PythonManager()

{

        Py_Finalize();

}



OP_API  int          PY_SayHello()

{
```

```
	PythonManager pm;


	PyObject* pModule;
	PyObject* pFunc;
	//call python function, with no parameters
	int nRet = 0;
	pModule = PyImport_ImportModule("myLib");
	if ( NULL == pModule )
		return nRet;


	pFunc = PyObject_GetAttrString(pModule, "SayHello");
	if ( pFunc )
	{
		PyObject* pRet = NULL;
		pRet = PyEval_CallObject(pFunc, NULL);


		PyArg_Parse(pRet, "i", &nRet);
	}
	return nRet;
}


static	float	_call_float_float(LPCSTR lpcszFunc, float a, float b)
{
	PythonManager pm;


	PyObject* pModule;
```

```
        PyObject* pFunc;
        //call python function, with multiple parameters
        float fRet = 0;
        pModule = PyImport_ImportModule("myLib");
        if ( NULL == pModule )
                return fRet;

        pFunc = PyObject_GetAttrString(pModule, lpcszFunc);
        if ( pFunc )
        {
                PyObject* pParams = NULL;
                pParams = PyTuple_New(2);             //create tuple to put
parameters

                PyTuple_SetItem(pParams, 0, Py_BuildValue("f", a));
                PyTuple_SetItem(pParams, 1, Py_BuildValue("f", b));
                PyObject* pRet = NULL;
                pRet = PyEval_CallObject(pFunc, pParams);

                PyArg_Parse(pRet, "f", &fRet);
        }
        return fRet;
}
OP_API  float PY_Add(float a, float b)
{
        return _call_float_float("Add", a, b);
}
```

```
OP_API  float PY_Sub(float a, float b)
{
        return _call_float_float("Sub", a, b);
}

OP_API  float PY_Mult(float a, float b)
{
        return _call_float_float("Mult", a, b);
}

OP_API  float PY_Div(float a, float b)
{
        return _call_float_float("Div", a, b);
}

OP_API  float PY_Mod(float a, float b)
{
        return _call_float_float("Mod", a, b);
}
```

Add a Module-Definition File, named OPython.def, to the project, and replace the original code with the code below.

```
LIBRARY  "OPython"

EXPORTS
```

```
;Functions to export
PY_SayHello
PY_Add
PY_Sub
PY_Mult
PY_Div
PY_Mod
```

Build the solution to generate the DLL, OPython.dll, by default

20.2.4.4 Use the DLL

The following code show how to use the DLL generated above in Origin C.

Copy the DLL generated above (OPython.dll), and paste it to the directory where Origin installed.

Copy the header file created above (OPython.h) to this folder: **<User Files Folder>/OriginC/**.

Start 32-bit version of Origin and launch Code Builder. Create a new C file and replace with the following code to it.

```
#include <Origin.h>
#include "OPython.h" //remember to include this header.

int test_Python()
{
        int nRet = PY_SayHello();

        float c = PY_Add(2, 4);

        float d = PY_Div(2, 3);
```

```
        return 0;
}
```

Compile the above code in Code Builder, and then run the following line in **LabTalk Console** (Open by Code Builder menu View: LabTalk Console).

```
test_Python;
```

20.2.4.5 Remarks

The example shown in this page is only to create a simple DLL that can reuse the functions written in Python. If to process a large amount of data in Python, and transfer the data between Origin and Python, a buffer will be recommended. With buffer, when transfer data from Origin's Column, the vector, which contains data, can be passed from Origin C to a DLL function, which receive a pointer (maybe double *pBuffer) as parameter. And any change based on this buffer will take effect immediately on the vector in Origin C. In **\Samples\Python** folder of your Origin software, you can see code example(CCallPython.cpp , CCallPython.py , and python.h) to transfer data buffer by two OC API:

```
OPYTHON_API OIP OPy_GetItemsFromList(int nType, void* pyObj, void* pBuffer);

OPYTHON_API void* OPy_SetItemsToList(int nType, void* pBuffer, OIP* pnDims, int
nDims);
```

20.3 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 Server and Client\MS Office\Client* 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 Server and Client\ADO\Client* subfolder of Origin.

21 Reference

21.1 Reference

This section covers the following topics:

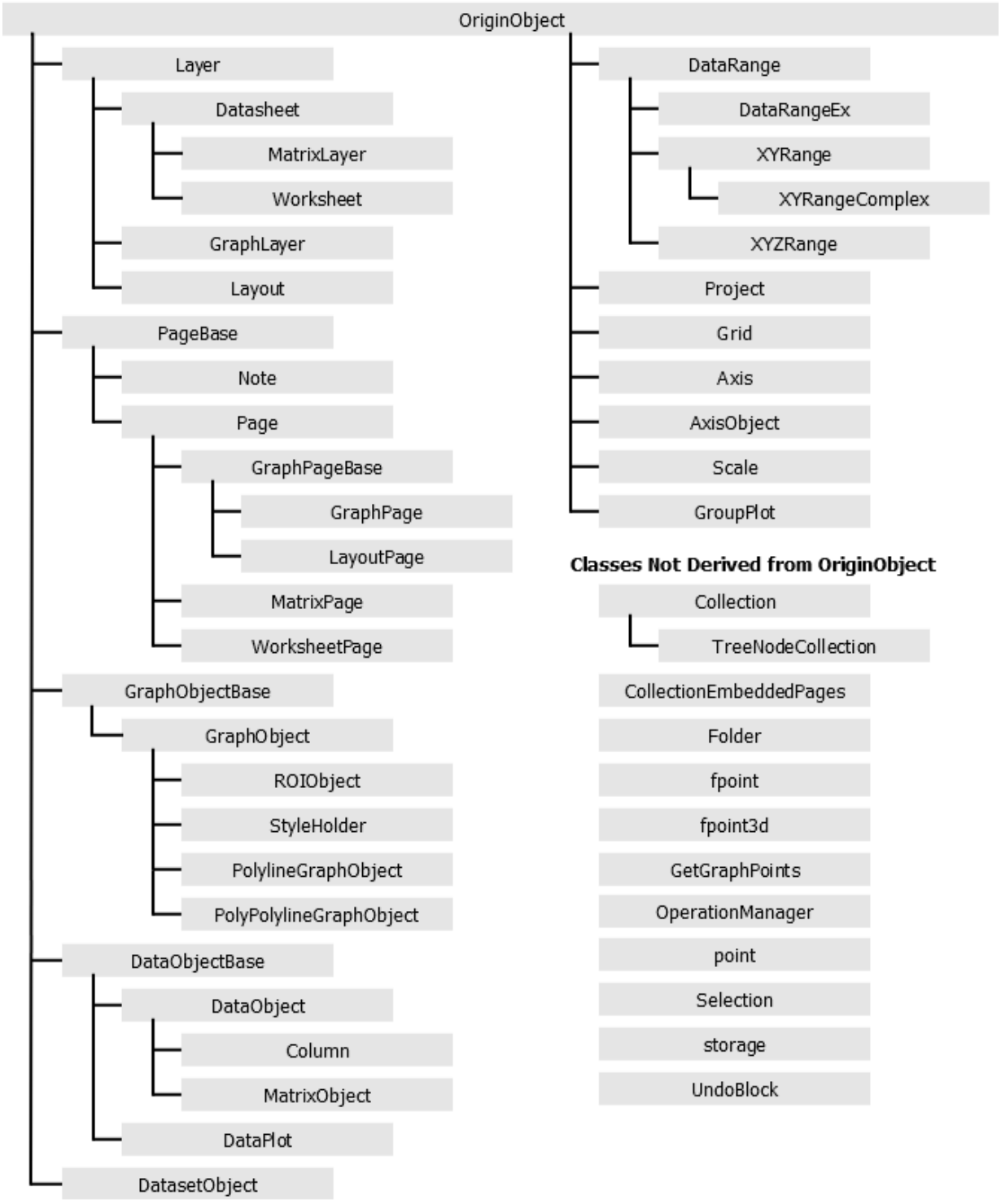
Class Hierarchy

Collections

21.2 Class Hierarchy

The following schematic diagram shows the hierarchy of built-in Origin C classes.

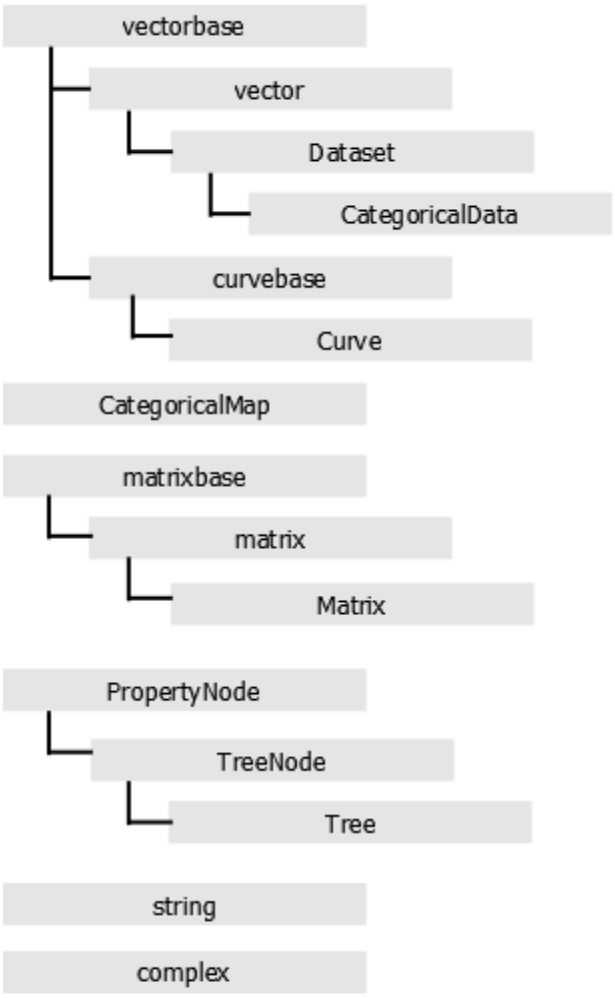
Internal Origin Objects



Classes Not Derived from OriginObject

- Collection
  - TreeNodeCollection
- CollectionEmbeddedPages
- Folder
- fpoint
- fpoint3d
- GetGraphPoints
- OperationManager
- point
- Selection
- storage
- UndoBlock

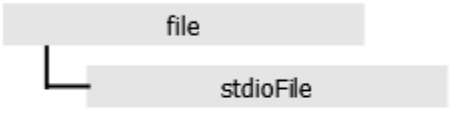
Composite Data Types



Utility

Array
BitsHex
JSON
OrgAppInfo
OrgApps
Profiler
VideoWriter

System



INIFile
Registry

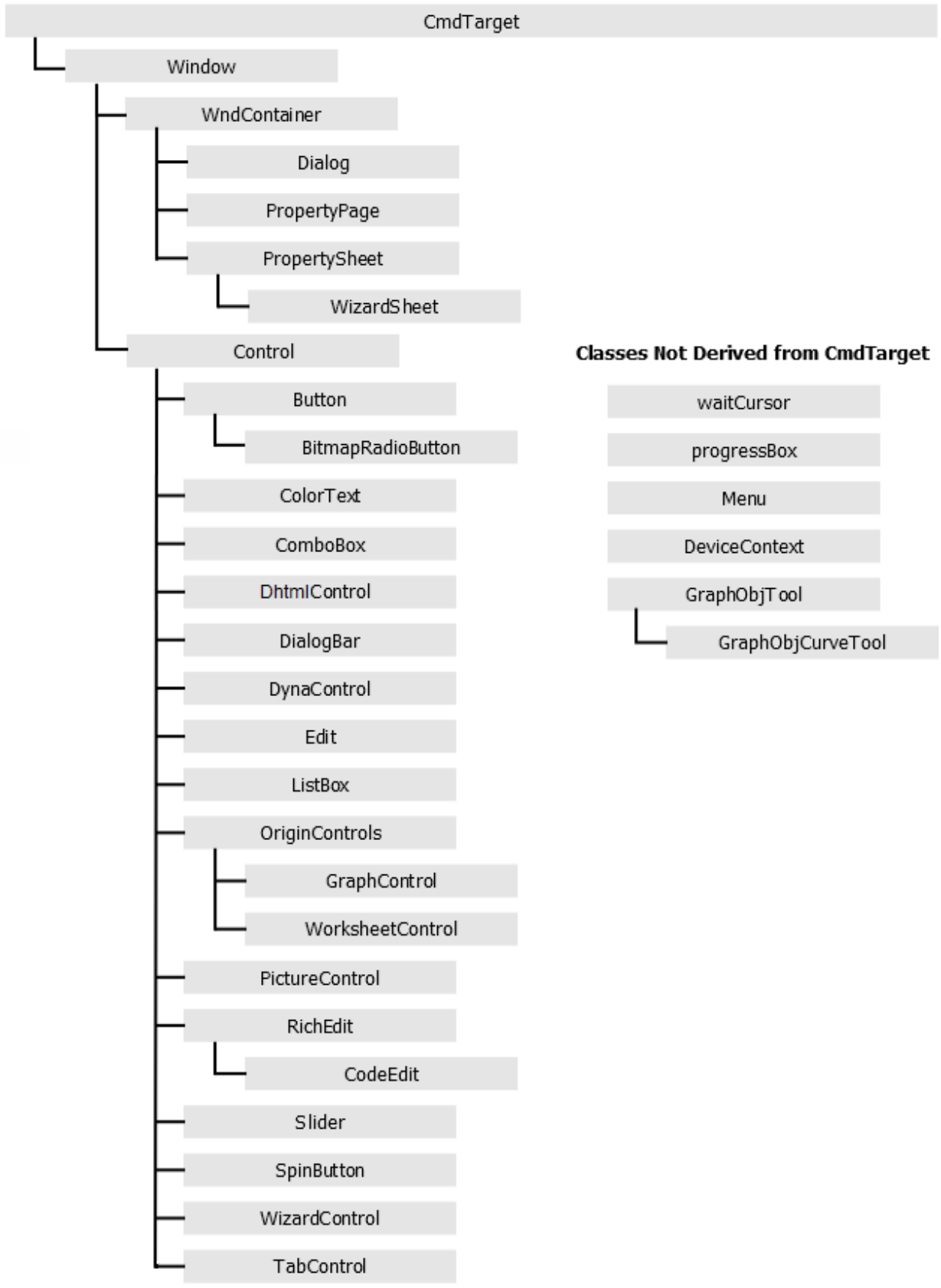
Application Communication

Matlab

Analysis

NLFitSession
NLFitContext

User Interface Controls



Classes Not Derived from CmdTarget

- waitCursor
- progressBox
- Menu
- DeviceContext
- GraphObjTool
  - GraphObjCurveTool

21.3 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);
}
```

Index

A 22

Accessing X-Function 303

Active Column 108

Active Matrixbook 67

Active Page, Page Active 203

Active Window 67

Active Workbook 67

Active Worksheet 108

Add Column 96

Analysis Functions 235

Arithmetic 100

Arithmetic Operators 8

Array, One-Dimensional 187

Array, Two-Dimensional 6–5

Array, Two-Dimensionals 188

Auto Update 472

B

Baseline, Create 273

Baseline, Remove 273

Binning, 2D 247

break Statement 18

Building by Script 50

Building on Startup 49

Building Origin C File with Project 48

Building Origin C Files under Workspace System Folder 48

Building with Dependent Files 50

Building, Handle Compiling and Linking Error 50

C

C# DLL 477

C# Features 1

C++ Features 1

C++(.Net) DLL 477

Caret Character 9

char Type 5

Class Hierarchy 509

Classes 24

Classes, Analysis Category 29

Classes, Application Communication Category 30

Classes, Data Types Category 30

Classes, Origin Objects Category 31

Classes, Predefined 24, 29

Classes, System Category 33

Classes, User Defined 24, 333, 340

Classes, User Interface Controls Category 34

Classes, Utility Category 37

Clone Page 92

Code Builder 1, 46

Collections 513

Color Palette 6

Color, Auto Color 7

Color, Check as RGB Value or Palette Index 7

Color, Extract Single Color from RGB 8

Column Designation 99

Column Label 97

Column Properties 99

Column, Access Data Buffer 119

Column, Add Tree Variable 208

Column, Data Format 100

Column, Data SubFormat 100

Column, Get the Names of All Storages 209

Column, Get Tree Variable 208

Column, Get/Set Date and Time Data 104

Column, Get/Set Numeric Data 103

Column, Get/Set String Data 104

Column, Setting Value by Formula 101

COM 505

Comments Label 97

Comparison Operators 10

Compile and Link Source Files 47

Compiler, Generate Pre-processed File 41

Compiler, Type of Pre-processed File 41

Complex Variable 190

Complex, Imaginary Component 6

Complex, Real Component 6

continue Statement 18

Contour Graph, Get/Set Z Levels of Colormap 163

Contour Graph, Set Line Style 168–67

Convolution 270

Copy Format 160

Copy Matrix 81

Correlation 270

Correlation Coefficient 247

Curve Fitting 261

D

Data Comparison 128

Data Range Control 192

Data Range, from Graph 192

Data Range, from Matrix 191

Data Range, from Worksheet 191

Data Reader 382

Data Selector 192

Data Types 5

Database 291

Database, SQL 507

DataPlot, Bar/Column 150

DataPlot, Contour 151

DataPlot, Image Type 153

DataRange, Construct XYZ Range 129

Date Time, Convert Julian Date to String 198

Date Time, Convert String to Julian Date 198

Date Time, Custom Date Format 198

Date Time, Get Current Date Time 197

Debugging 51

Debugging, Breakpoints 51

Delete Column 97

Delete Worksheet 108

Descriptive Statistics 246

Dialog Boxes 312

Dialog Builder 324, 325

Dialog with Tree View Control 339

Dialog, Event Functions 346

Dialog, Open 348

Dialog, Resource DLL Created by Origin Dialog AppWizard 325

Dialog, Resource DLL Created by Visual Studio 2008 326

Dialog, Resource DLL Created by Win32 Dynamic-Link Library 326

Differentiation 245

Distributing Code 59

double Type 5

Duplicate Window 65, 92

E

Exception Handling 26

Exponentiate Operator 9

Export Data from Matrix to ASCII File 231

Export Data from Worksheet to ASCII File 229

Export Data to Database 293

Export Graph to Image File 230

Export Image from Matrix to Image File 232

External Application, Database 507

External Application, Execute Matlab Command 505

External Application, MS Excel 505

External Application, MS Word 505

External DLL, Access 477

External Resources, Access 477

Extract Frame 225

Extrapolation 237

F

FDF File 266

FFT 270

FFT Filtering 270

File Open Dialog 315

File SaveAs Dialog 316

File, Object File(*.OCB) 40

File, Preprocessed File(*.OP) 41

File, Workspace File(*.OCW) 42

float Type 5

Flow Control Statements 13

Folder, Get Path 200

Folder, Get Sub Folders 201

Folder, Move Folder 202

Folder, Move Window 202

Font, Convert from Index to DWORD 169

for Statement 16

foreach Statement 19, 203

Fortran 477

Frequency Count 246

Functions, Adding for Set Values Dialog F(x) menu 57

Functions, Calling Origin C Function from Script 53

Functions, Control LabTalk Access 53

Functions, Execution 3, 46, 53

Functions, List All LabTalk Callable Origin C Functions 54

Functions, Pass LabTalk Variable to Origin C by Reference 54

Functions, Pass LabTalk Variable to Origin C by Value 54

Functions, Precedence Rules for the Same Name Functions 57

Functions, Predefined 20

Functions, User-defined 21

G

Gadget Tool 431

GetN Data Selector Control 192

GetN Graph Preview Dialog 404

GetN Simple Dialog 316

GetN, Image Dialog 409

Global Functions 20

goto Statement 19

Graph Browser 461

Graph Layer, Add 3D Surface Plot 150

Graph Layer, Add Data Marker 157

Graph Layer, Add Worksheet Table 183

Graph Layer, Add XY Data Plot 148

Graph Layer, Axis Label Properties 144

Graph Layer, Axis Properties 144

Graph Layer, Axis Tick Label Properties 146

Graph Layer, Axis Ticks Properties 146

Graph Layer, Border 141

Graph Layer, Copy 110

Graph Layer, Dimension 176

Graph Layer, Double-Y Axes 153

Graph Layer, Fill Color 141

Graph Layer, Get Active Data Plot 149

Graph Layer, Get Format 141

Graph Layer, Get Format on DataPlot 158

Graph Layer, Grid Lines 142

Graph Layer, Legend 183

Graph Layer, List all Data Plots 514

Graph Layer, Refresh 145

Graph Layer, Remove Data Marker 157

Graph Layer, Rescale 148

Graph Layer, Scale Settings 143

Graph Layer, Set Color on DataPlot 158

Graph Layer, Set Data Marker 157

Graph Layer, Set Format on Grouping Plot 162

Graph Layer, Set Format on Line Plot 159

Graph Layer, Set Format on Scatter Plot 161

Graph Layer, Show Additional Lines 142

Graph Layer, Show/Hide Axis 145

Graph Object, Add Curve Arrow 179

Graph Object, Add Rectangle 177

Graph Object, Add Text Label 177

Graph Object, Attach Property 181

Graph Object, Create 177

Graph Object, Disable Moving and Selection 181

Graph Object, General Properties 180

Graph Object, Position 180

Graph Object, Programming Control 182

Graph Object, Resize 180

Graph Object, Set Line Position 178

Graph Page, Add Layer 172

Graph Page, Arrange Layers 154

Graph Page, Attach a User Defined Dialog to It 383

Graph Page, Delete 139

Graph Page, Get Format 140

Graph Page, Get the Existing Window 139

Graph Page, Gradient Control 141

Graph Page, Layer Arrangement 170, 173

Graph Page, Linking Layers 176

Graph Page, Move Layer 173

Graph Page, Plotting Separated Data in Multiple Layers 171

Graph Page, Rename 139

Graph Page, Resize Layer 174

Graph Page, Swap Layers 175

Graph Preview Dialog 333

Gridding 128

H

Hello World Dialog 325, 350

Hello World Function 1

Hidden Page, Create 90

Hide Column 98

HTML Dialog 349, 350, 354, 364

I

if Statement 13

if-else Statement 13

Import ASCII Data File to Matrixsheet 218

Import ASCII Data File to Worksheet 215

Import Data with Filter 218

Import File, Get Storage Tree by Name 209

Import Files with Import Wizard 221

Import Image into Matrix 222

Import Image into Worksheet Cell 224

Import Image to Graph 225

Import Video 225

Importing from a Database 291

Importing, Get Info from Image File 223–22

Improve OC Execution Speed 119

Input String 312

Insert Column 96

Integer Type 5

Integral, Multi-dimension 243

Integral, with Multiple Parameters 240

Integral, with One Integration Variable 238

Integration 238

L

LabTalk Variable, Get/Set in Origin C 297

LabTalk Variable, Get/Set in Origin C, LabTalk Object, Use in Origin C 300

LabTalk Variable, LabTalk Script, Execution in Origin C 299

Language Basic Features 1

Language Fundamentals 5

Layer, Alignment 175

Linear Fitting 249

List Labtalk Callable Functions 54

Logical Operators 11

Long Name Label 97

loop Statement 16

LT_execute 299

LT_get_str 299

LT_get_var 297

LT_set_str 299

LT_set_var 297

M

Mathematics 235

matrix 6

Matrix 188

Matrix Object, Set Value By Formula 80

matrix sheet 75

Matrix Sheet, Copy 110

Matrix Sheet, Data Format 80

Matrix Sheet, Set Internal Data Type 79

matrix, copy 81

Matrix, Dot Multiply 82

Matrix, Multiply with Constant 81

Matrixbook, Activate a Sheet 67

Matrixbook, Create 90

matrixbook, Get Name 67

matrixbook, Get One Sheet 67

Matrixsheet, Get/Set Labels for XY 70

Matrixsheet, Get/Set Labels for Z 72

Message Box 313

Message Map 340

Missing Values 185

Modal Dialog 330

Modulus Operator 10

Move Column 98

Multiple Linear Regression 259

Multiple XY Datasets 472

N

NAG Callback Functions 278

NAG Error Structure 278

NAG Functions 277

NAG Get Data From Origin 279

NAG Header Files 277

NAG, Memory Free 280

NANUM 185

New a Source File 46

NLFit, Get Number of Parameters 262

NLFit, Set Parameter Values 263

NLFitSession Class 261

NLFitSession, Get Fitted Data 264

NLFitSession, Get Number of Iterations 264

NLFitSession, Get Reduced Chisqr 264

NLFitSession, Run Code to Initialize Parameter Values 263

NLFitSession, Set Fitting Data 262

NLFitSession, Set Fitting Function 262

NLFitSession, Share Parameters 264

NLFitSession, Statistic Result 264

Nonlinear Fitting 261

Normality Test 248

Normalize 235

Numeric, Convert to String 187

Numeric, Data Comparison Data 186

O

Operation, List All 211

Operators 8

Origin Object, Copy Format 160

Origin System Path 197

Output to Notes Window 288

Output to Report Sheet 289

Output to Results Log 287

Output to Script Window 287

oxf File 449

P

Package Files 59

Page, Clone 65, 92

Page, Get Name 139

Page, Get Path 201

Palette 6

Parameters Label 97

Path Browser Dialog 316

Peaks, Find 275

Peaks, Fit 276

Peaks, Verify Peaks by Minimum Height 276

Pick Data Points 382

Plotting in Multiple Graph Layers 154

Plotting, from Matrix 150

Plotting, from Virtual Matrix 135

Plotting, from Worksheet 148

Pointer 5

Polynomial Fitting 257

pragma 477

Progress Box 314

Project, Add Range Variable 205

Project, Add Tree Variable 205

Project, Append 199

Project, Get Active Folder 200, 201

Project, Get the Active Layer 203

Project, Get the Active Page 203

Project, Get the Existing Window 202

Project, Get the Names of All Tree Variables 206

Project, Get Tree Variable 206

Project, List all Graph Pages 514

Project, List all Matrixbooks 514

Project, List All Operations 211

Project, List All Range Variables 205

Project, List all Workbooks 514

Project, Loop the Existing Windows 203

Project, Modified Flag 200

Project, New 199

Project, Open 199

Project, Save 199

R

Recalculation 465

Rename Column 97

Report Sheet, Get Result Tree 210

Report Sheet, Get the Specified Result 212–11

Report with Auto Update 472

Result Sheet 472

RGB 7

S

Script Window 287

Select Data Range 465

Set Column Values 101

Set Formula 101

Share Code 59

short Type 5

Signal Processing 270

Smoothing 270

Source File, New 339

Sparkline 99

SQL 291

SQLite 295

Statistics Functions 246–45

STFT 270

string 6

String Array 6

String, Append Numeric 194

String, Convert to Complex 194

String, Define Variables 193

String, Extract File Name from Full Path 196

String, Get Sub String 195

String, Replace 195

Swap Column 98

switch Statement 15

T

Template 140

Theme File 160

throw Statement 26

TreeNode 189

try-catch Statement 26

U

Units Label 97

User Defined Labels 97

User File Folder 197

User Interface 311

Using NAG Functions 277

V

Variable Extraction in Import Wizard 222

vector 6

Vector 187

Video 225

Video Writer 232

Virtual Matrix 135

void Return Type 5

W

Wait Cursor 381

Wavelet 270

while/do-while Statement 17

Wizard Dialog 420

Wizard Dialog, Add Steps Map 331

WizardSheet 331

Workbook, Activate a Sheet 107

Workbook, Delete One Sheet 108

Workbook, Get Name 108

Workbook, Get One Sheet 108

Workbook, Get Page from Layer 109

Workbook, Reorder Sheets 109

worksheet 118

Worksheet 97

Worksheet, Add Tree Variable 206

Worksheet, Alignment 111

Worksheet, Check If Hierarchy 211

Worksheet, Compare Data 127

Worksheet, Convert Data To Matrixsheet 128

Worksheet, Delete All Columns 119

Worksheet, Embedded Graph 120

Worksheet, Extract Data 125

Worksheet, Find Data 122

Worksheet, Get Data from Specified Data Range 118

Worksheet, Get Sheet Name 109

Worksheet, Get the Names of All Storages 207

Worksheet, Get Tree Variable 207

Worksheet, Highlight Selection 125

Worksheet, Merge All Sheets 110

Worksheet, Merge Cells 113

Worksheet, Put Text to Cell 111

worksheet, reduce data 124

Worksheet, Set Color for the Selected Range 126

Worksheet, Set Column Width by LabTalk Command 299

Worksheet, Set Display Range 118

Worksheet, Sorting Data 121

Worksheet, Text Font 111

Workspace, Load Files 47

Workspace, Packages Folder 43

Workspace, Project Folder 45

Workspace, System Folder 45

Workspace, Temporary Folder 45

Workspace, User Folder 45

X

XF_THROW macros 404

XF_TRACE macro 403

XF_WARN macros 403

XFBase Class 303, 454

X-Function Builder 387, 390

X-Function Custom Dialog 414

X-Function Dialog Controls 392

X-Function Dialog Theme 393

X-Function Option Strings 393

X-Function ReportData 472

X-Function ReportTree 465

X-Function Tree View 393

X-Function Variable 390

X-Function Variable Type 391

X-Function, Add Custom Button on Dialog 449

X-Function, Add Documentation 449

X-Function, Add Internal Functions 396

X-Function, Add to Origin Menu 446

X-Function, Auto Compile Dependent Files 395

X-Function, before_execute 397, 427

X-Function, Coding in Code Builder 394, 406–5, 411

X-Function, Construct Controls on Dialog 457–56

X-Function, Control Labtalk Accessing 389

X-Function, Create 387, 388

X-Function, Create Graph Browser Dialog 461

X-Function, Display Error Message on Dialog 402

X-Function, Error Handling 402

X-Function, Event Handler Functions in Image Dialog 401

X-Function, Event Handler Functions in Preview Dialog 400

X-Function, Event Handling Functions 396

X-Function, event1 399, 450

X-Function, Execution 389, 413

X-Function, Execution from Script 453

X-Function, Execution in Origin C 454

X-Function, Execution with Theme 431

X-Function, Execution with Wizard Dialog 430

X-Function, Execution without Dialog 453

X-Function, Fitting 473

X-Function, Handle Runtime Error 404

X-Function, Image Dialog Init Event Function 412

X-Function, Image Dialog OnChange Event Function 412

X-Function, Include Header Files 395

X-Function, Main Function 396

X-Function, make_tree 399, 433

X-Function, Output Error Message 402

X-Function, Output Warning Message 403

X-Function, Set Parameters in Origin C 454

X-Function, Statistics 468

X-Function, Update Control Value on Dialog 461

XOR Operator 12