

Debugging with XLOPER

1. Steps

2. Source Code

This article provides source code for a macro which makes it easier to show the contents of an XLOPER array within the debug window. This can be useful when searching for data problems that relate to input tables.

This macro is designed to work as a debugging "framework" that uses the intelligence of the Visual C++ 6 IDE to output debug statements that allow you to navigate the code easily. This article is only of interest to C or C++ developers working on the Microsoft Windows platform who use the Visual Studio development environment. The main idea is that the macro will output data to the debug window when you are running the code in Debug mode but will do nothing when the code is compiled and run in release mode. The Debug compiler directive is used to control this behavior.

Please note that you *must* use Debug mode (or equivalently `#define _DEBUG`) and also ensure that *Debug Info* (Project->Settings->C++) is set to *Program Database* since Visual C++ 6 doesn't replace `__LINE__` macros properly if your debug info is generated using *Program Database for Edit and Continue*

Steps

Add `fc_trace.c` and `fc_trace.h` to the source directory for your Visual C++ project and include `fc_trace.c` within the project. The source for these files is included later in this article.

`#include "fc_trace.h"` to files you are debugging

`fc_trace.h` defines three macros:

- `printLPXLOPER(LPXLOPER)`
- `printDouble(double)`
- `printInt(int)`

Note: These macros must be used with variables, not constant values.

An `LPXLOPER` is a long pointer to the XLOPER data structure. The argument to `printLPXLOPER` is assumed to be a pointer to an array of the XLOPER data structure. The routine will print the contents of an array of XLOPER into the debug window. The `printDouble` and `printInt` routines provide the same functionality for `double` and `int` variables.

Intersperse calls to these within the code you are debugging. As you execute the code in debug mode it will printout the contents of the XLOPER array to the debug window.

Now let's look at a sample program that makes calls to the routines.

```
1 #include "fc_trace.h"
2
3 /*****
4  aaBond_AU_cf
5  *****/
6 void FCExampleBondAUcf()
7 {
8  double d_s;
9  double d_m;
10 double cpn;
```

```

11 double princ_m;
12 static XLOPER hl[10];
13 int d_rul;
14 int table_type;
15 LPXLOPER Return_cfx_tbl;
16
17 /* Initialize the input variables */
18 d_s = 37390;
19 d_m = 38523;
20 cpn = 0.06;
21 princ_m = 100;
22
23 hl[0].xltype = 64;
24 hl[0].val.array.rows = 9;
25 hl[0].val.array.columns = 1;
26 hl[0].val.array.lpparray = &hl[1];
27
28 hl[1].xltype = 1; hl[1].val.num = 36892;
29 hl[2].xltype = 1; hl[2].val.num = 36942;
30 hl[3].xltype = 1; hl[3].val.num = 36995;
31 hl[4].xltype = 1; hl[4].val.num = 37040;
32 hl[5].xltype = 1; hl[5].val.num = 37076;
33 hl[6].xltype = 1; hl[6].val.num = 37138;
34 hl[7].xltype = 1; hl[7].val.num = 37173;
35 hl[8].xltype = 1; hl[8].val.num = 37218;
36 hl[9].xltype = 1; hl[9].val.num = 37250;
37
38 d_rul = 1;
39 table_type = 1;
40
41 printLPXLOPER(hl); /* show holiday list in the debug window */
42
43 /* Call the function */
44 Return_cfx_tbl = aaBond_AU_cf(d_s, d_m, cpn, princ_m, hl, d_rul, table_type);
45 if ((Return_cfx_tbl->xltype == 16) || (Return_cfx_tbl->xltype == (16|FCbitDLLFree)))
46     MessageBox( NULL, "Function failed to calculate", "fincad", MB_OK);
47
48 printLPXLOPER(Return_cfx_tbl); /* show the cashflow table */
49
50 aaGlobalFreeAllA();
51 }
    
```

Source Code

Below is the `fc_trace.h` header file.

```

1 #ifndef _FC_TRACE_H_
2 #define _FC_TRACE_H_
3
4 #include <crtdbg.h>
5 #include <stdarg.h>
    
```

```

6 #include <stdio.h>
7 #include <string.h>
8
9 #include
10 #include
11
12 #ifdef __cplusplus
13 extern "C" {
14 #endif
15
16 /* function prototypes for macros */
17 void _printLPXLOPER(const char *s, LPXLOPER I);
18 void _printDouble(const char * s, double d);
19 void _printInt(const char *s, int t);
20 void _trace(const char *fmt,...);
21
22 #ifdef _DEBUG
23 // Define constants for print format
24 #define STR_1(line) #line
25 #define STR_2(line) STR_1(line)
26
27 #define printLPXLOPER(s) _printLPXLOPER(__FILE__ "(" STR_2(__LINE__) "): " #s, ##s)
28 #define printDouble(s) _printDouble(__FILE__ "(" STR_2(__LINE__) "): " #s , ##s)
29 #define printInt(s) _printInt(__FILE__ "(" STR_2(__LINE__) "): " #s , ##s)
30 #else
31 #define printLPXLOPER(...)
32 #define printDouble(...)
33 #define printInt(...)
34 #endif
35
36 #ifdef __cplusplus
37 }
38 #endif
39
40 #endif
    
```

Here is the source for the macro's that allow you to display the data in the debug window while tracing in debug mode (`fc_trace.c`).

```

1 #include "fc_trace.h"
2
3 /* prototype for function that adds data to debug window */
4 void _trace(const char *, ...);
5
6 /* prototypes for local functions */
7 void _printNum(LPXLOPER I);
8 void _printArray(LPXLOPER I);
9 void _printError(LPXLOPER);
10 void _printNil(LPXLOPER I);
11
12 #define _traceMAXSTRING 1024
    
```

```

13
14 /* prints to debug window in specified format */
15 void _trace(const char* format,...)
16 {
17     /* buffer used for generating debug line - thread safe */
18     char _buf[_traceMAXSTRING];
19
20     va_list args;
21     va_start(args,format);
22
23     _vsnprintf(_buf,
24               _traceMAXSTRING,
25               format,
26               args);
27     va_end(args);
28
29     _RPT0(_CRT_WARN,_buf);
30 }
31
32 void _printInt(const char * s, int t)
33 {
34     _trace("%s: (int) %d\n",s,t);
35 }
36
37 void _printDouble(const char * s, double d)
38 {
39     _trace("%s: (double) %f\n",s,d);
40 }
41
42 void _printNum(LPXLOPER l)
43 {
44     _trace(" (FctypeNum) %f\n",l->val.num);
45 }
46
47 void _printArray(LPXLOPER l)
48 {
49     int rows = l->val.array.rows, cols = l->val.array.columns,i,j;
50
51     _trace(" (FctypeMulti) %d rows, %d columns\n",rows,cols);
52
53     for (i = 1; i <= rows; i++)
54     {
55         for (j = 1; j <= cols; j++)
56         {
57             _trace("%f, ", (l->val.array.lpparray+((i-1)*cols)+(j-1))->val.num);
58         }
59         _trace("\n");
60     }
61 }
62
63 void _printError(LPXLOPER l)

```

```

64 {
65   _trace(" (FCTypeError) ");
66 }
67
68 void _printNil(LPXLOPER l)
69 {
70   _trace(" (FCtypeNil) ");
71 }
72
73 void _printLPXLOPER(const char * s, LPXLOPER l)
74 {
75   _trace("%s ",s);
76   if(!l)
77   {
78     _trace("LPXLOPER is null\n");
79     return;
80   }
81   switch(l->xltype)
82   {
83     case FCtypeMulti: /* 64 */
84       _printArray(l); break;
85     case FCtypeNum: /* 1 */
86       _printNum(l); break;
87     case FCtypeErr: /* 16 */
88       _printError(l); break;
89     case FCtypeNil: /* 256 */
90       _printNil(l); break;
91     default: break;
92   }
93   _trace("\n");
94 }

```

Disclaimer

Your use of the information in this article is at your own risk. The information in this article is provided on an "as is" basis and without any representation, obligation, or warranty from FINCAD of any kind, whether express or implied. We hope that such information will assist you, but it should not be used or relied upon as a substitute for your own independent research.