Calling C++ Developed DLL and XLL by Excel (2)

2007/08/13 羅紹玫

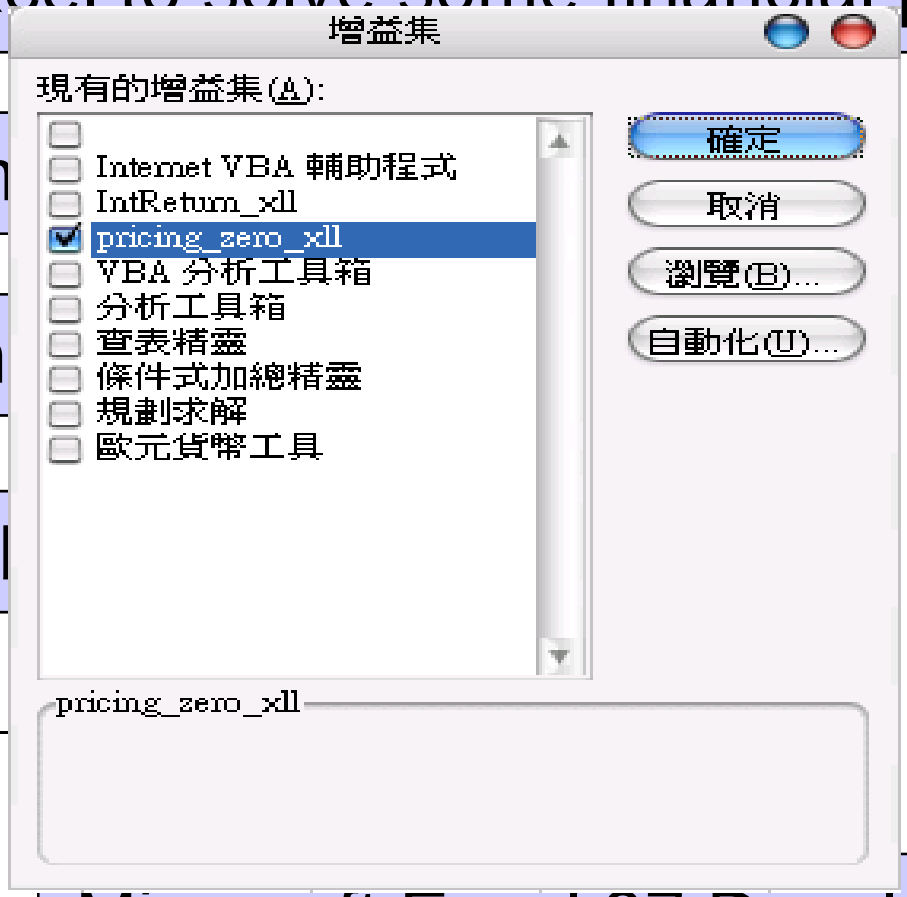
Review

Using Excel to solve some financial problems

function

Dynam

Extensil



Microsoft Excel 97 Developer's Kit

Turning DLLs into XLLs

- Creating XLL using Visual C++ 6.0

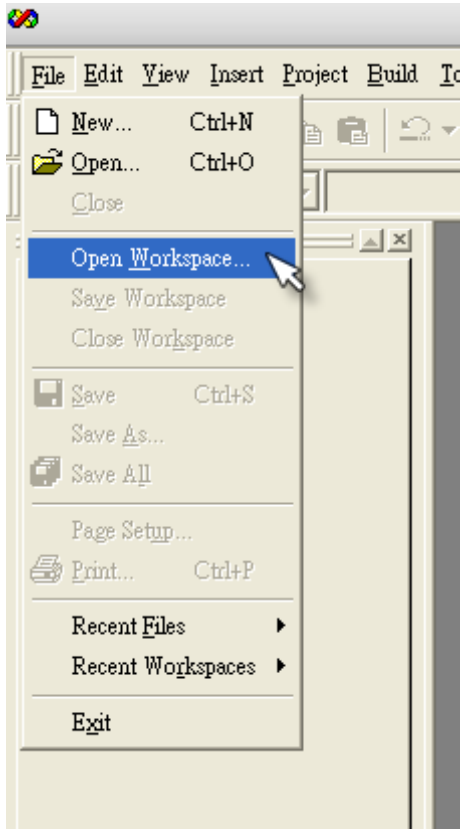
  -Download Excel 97 SDK

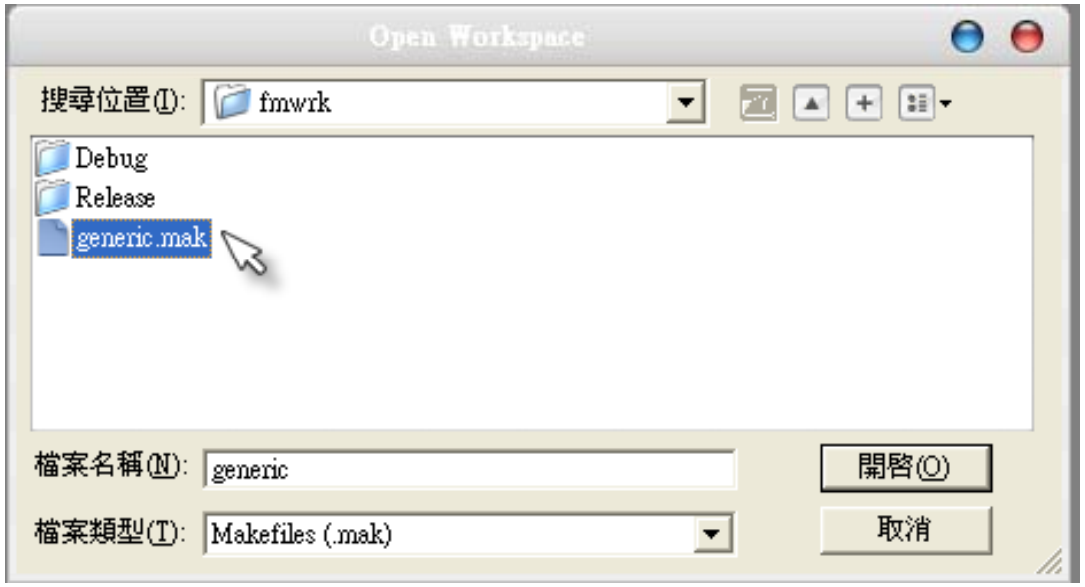
  -Framework (a template of XLL)

- Accessing XLL by Excel Add-in

Creating XLL using VC6

- Open a template project





Creating XLL using VC6 (Cont.)

- The xlAuto Interface function

-xlAutoOpen
  - Called when Excel starts up or adds-in loaded

-xlAutoClose
  - Called when Excel close down or adds-in unloaded

-xlAutoAdd
  - called when the Add-In Manager adds an XLL

-xlAutoRemove
  - called when the Add-In Manager removes an XLL

Creating XLL using VC6 (Cont.)

- The xlAuto Interface function (cont.)

-xlAddInManagerInfo

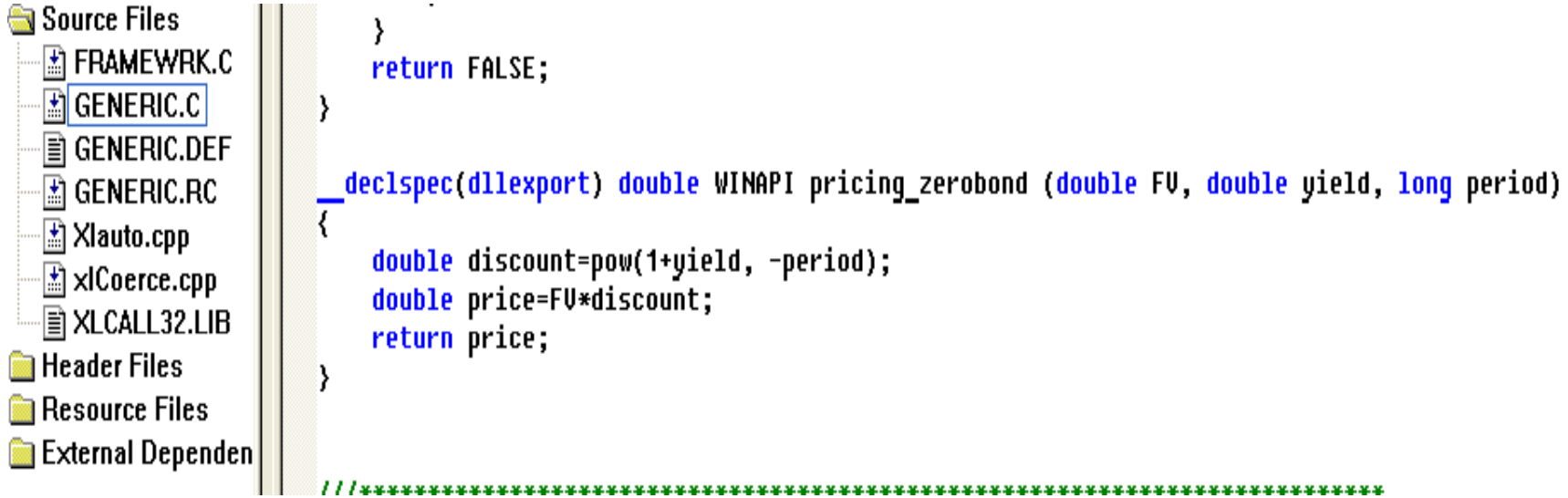
  - Called when the first time the Add-In Manager is invoked

-xlAutoRegister

  - called if a macro sheet tries to register a function without specifying the argument and return value types

Creating XLL using VC6 (Cont.)

- Adding code (pricing a zero-coupon bond)
  -in generic.c

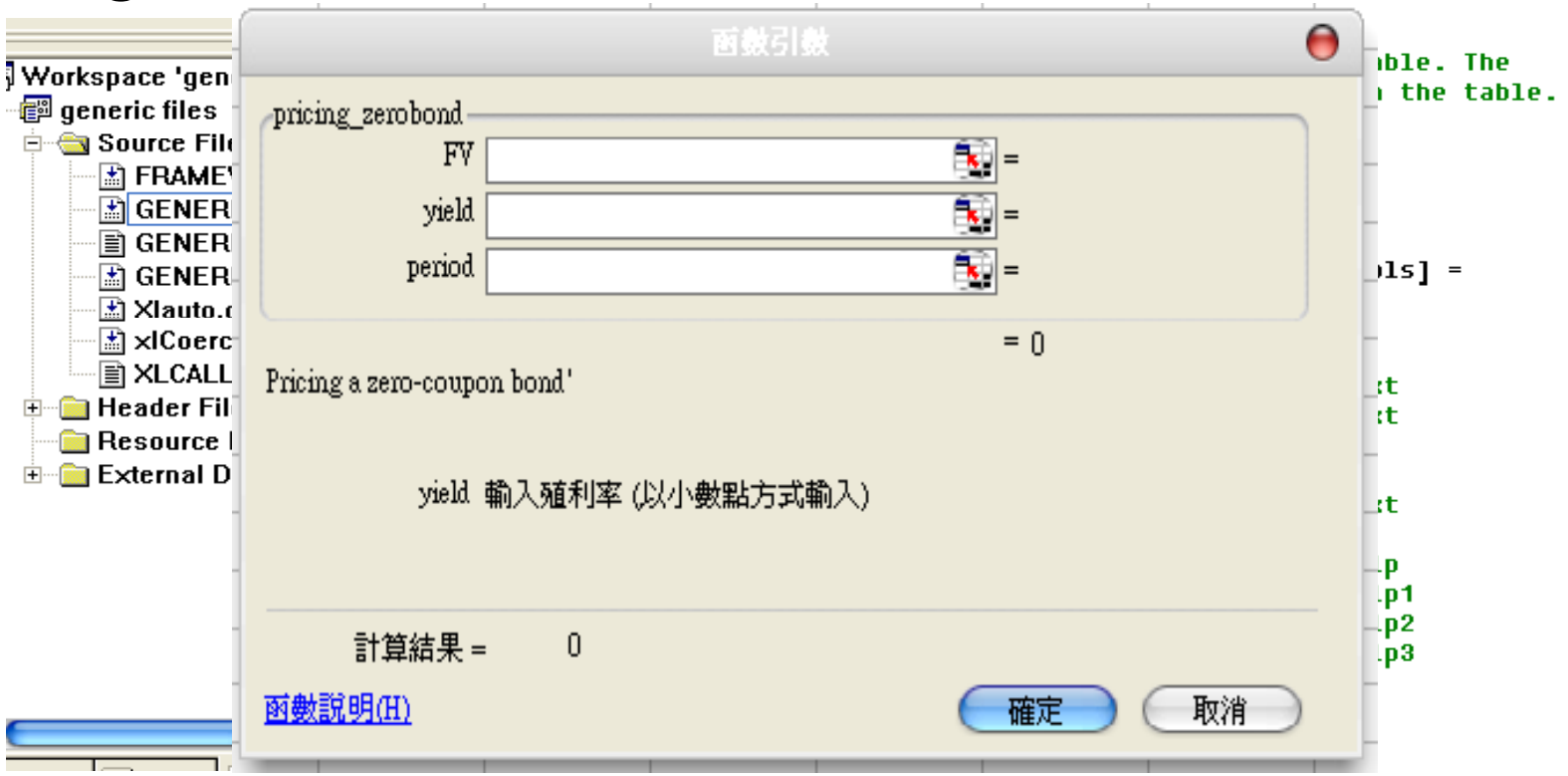


使用 **_declspec(dllexport)** 關鍵字匯出
DLL的資料、函式、類別或類別成員函式

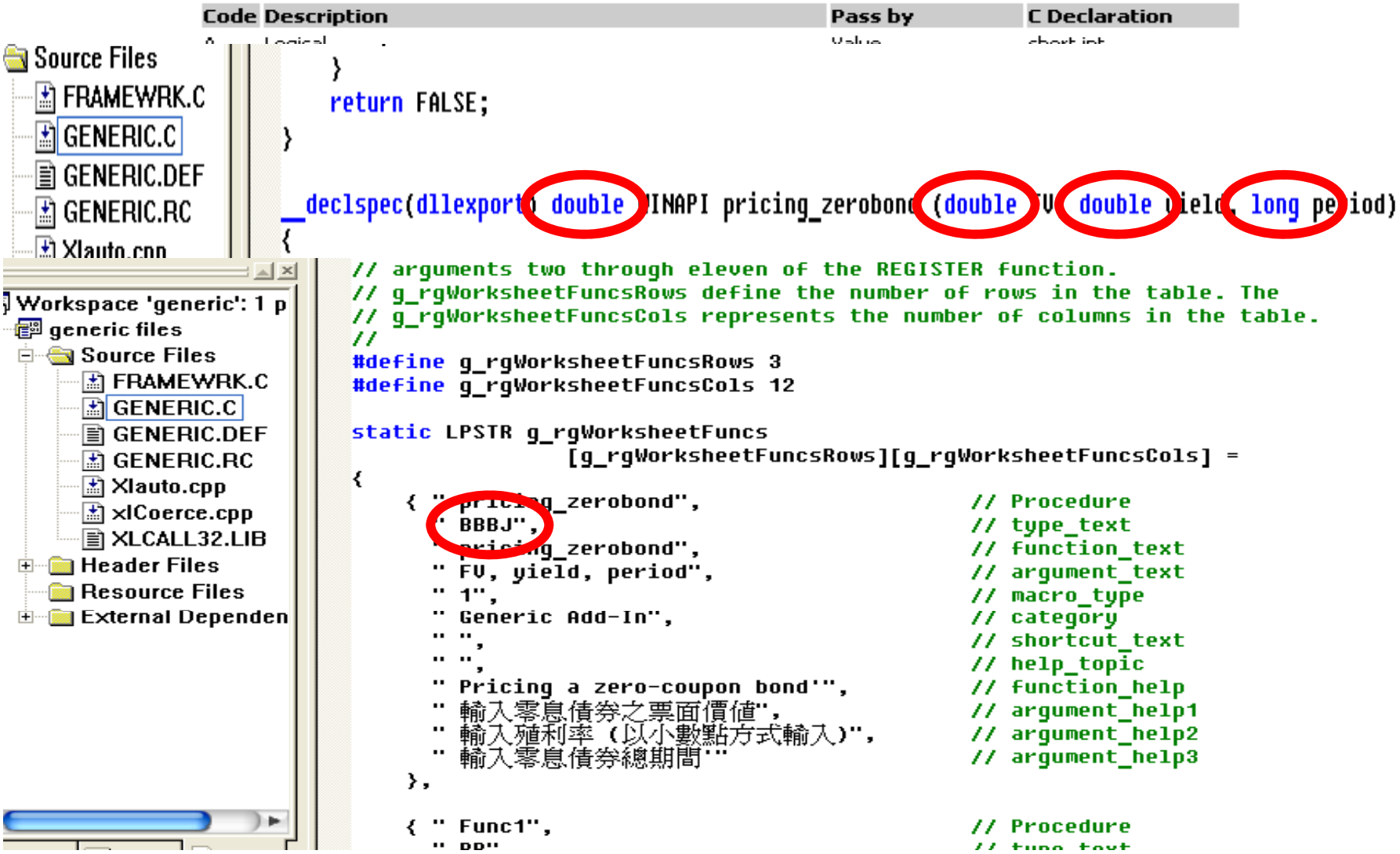
Creating XLL using VC6 (Cont.)

- Adding code

  -in generic.c



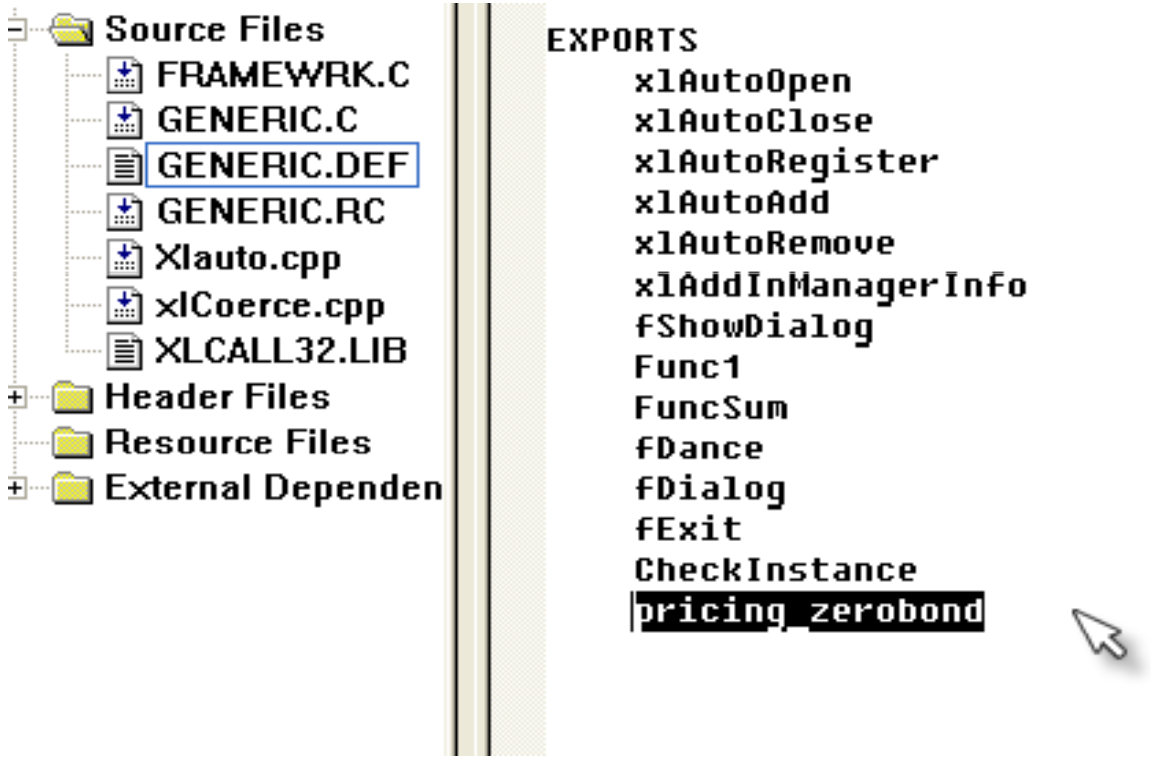
Creating XLL using VC6 (Cont.)



Creating XLL using VC6 (Cont.)

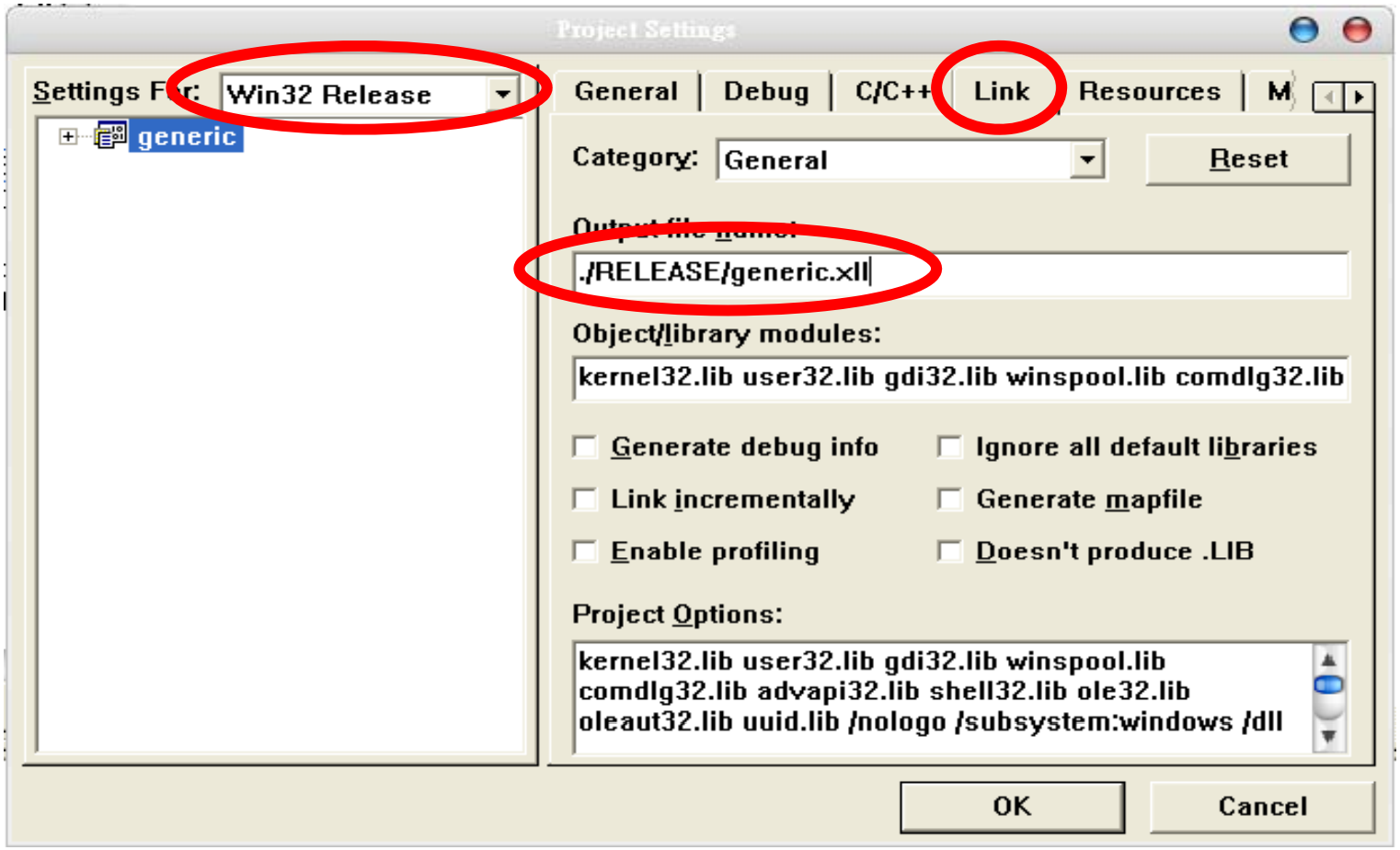
- Adding code

  -in generic.def



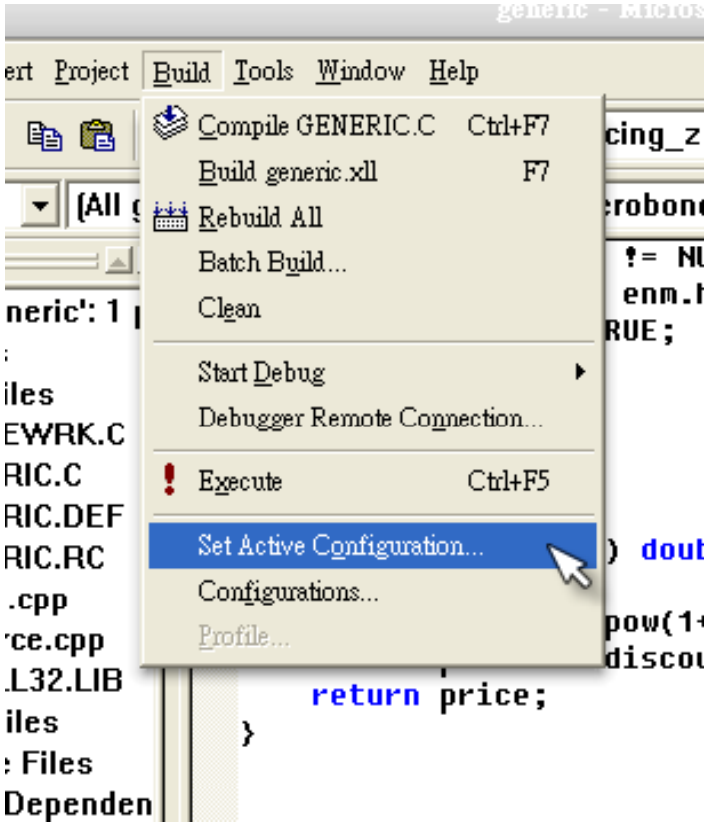
Creating XLL using VC6 (Cont.)

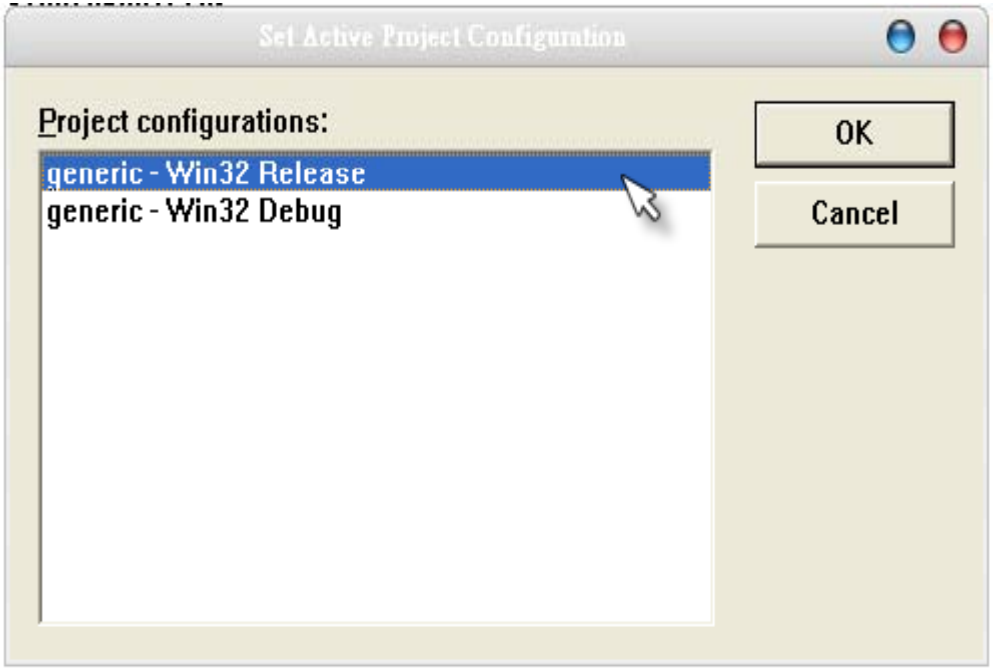
- Project / Settings (Alt+F7)



Creating XLL using VC6 (Cont.)

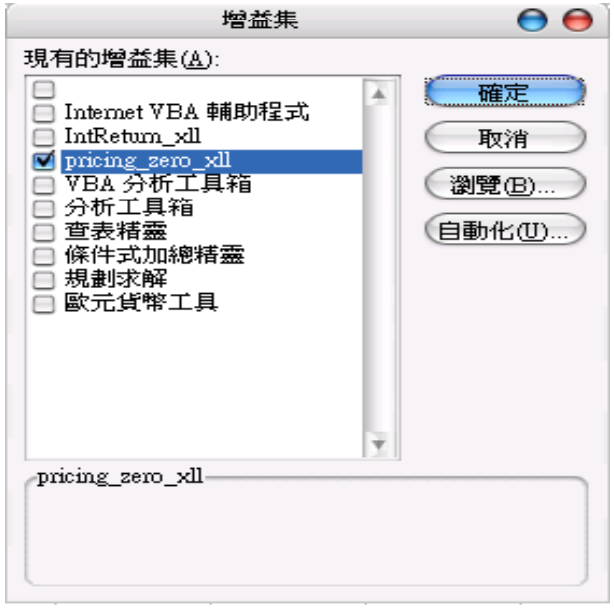
- Build the xll file

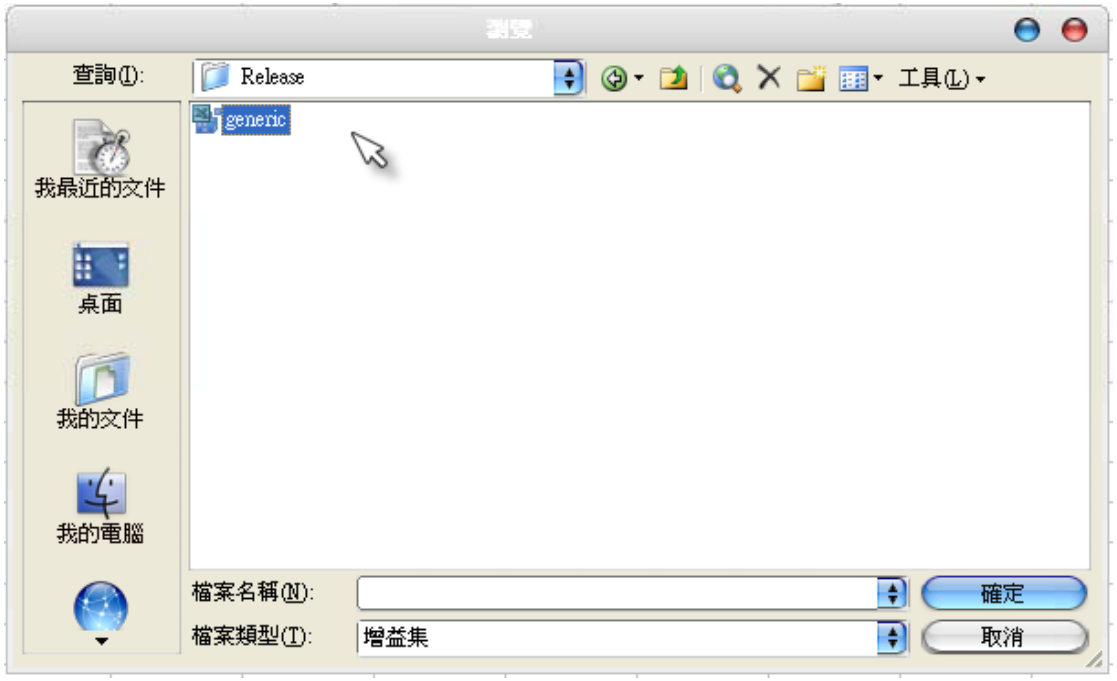




Accessing XLL by Excel

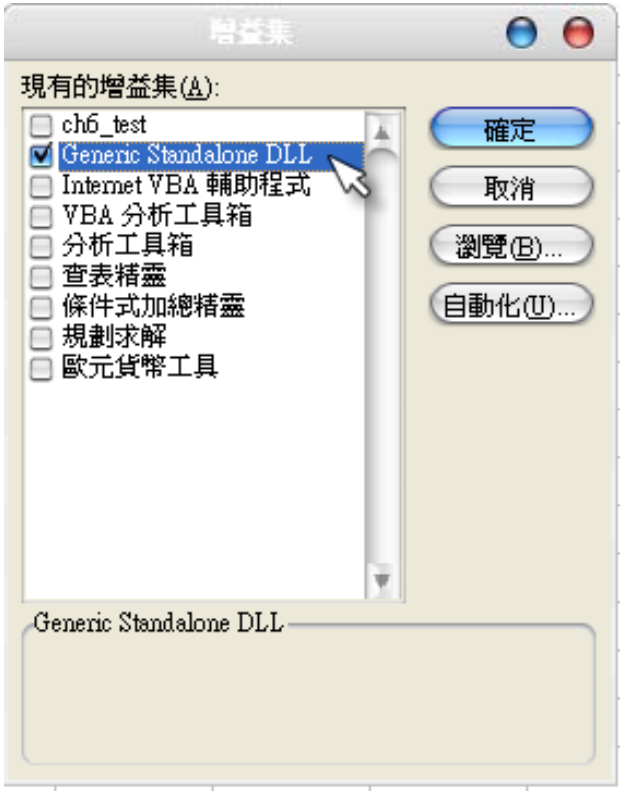
- In Excel

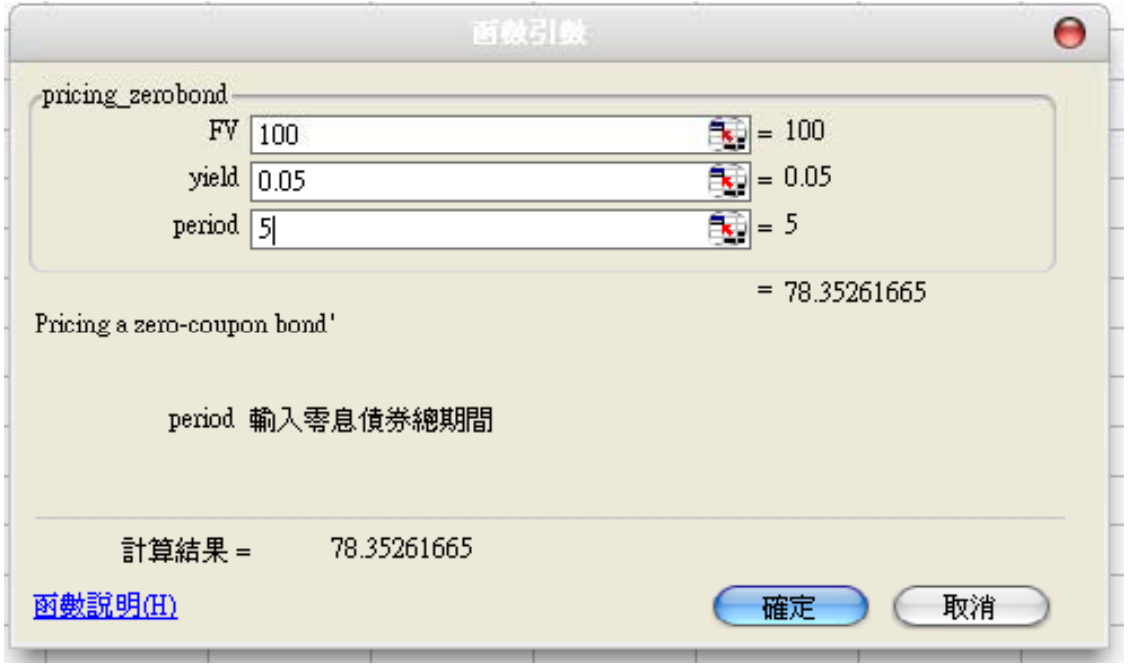




Accessing XLL by Excel (Cont.)

- Result





Pass Data between Excel & DLL

- Four ways to communicate

  - Via native C/C++ data types, converted automatically by Excel
  - Via a structure that describes and contains 2-D arrays of 8-byte doubles (xl_array)
  - Via a structure that can not only represent the contents of any cell, but also ranges and a few other things (xloper)
  - Via a structure that can represent the contents of any cell (oper)

Native C/C++ data types

- [signed] short [int] (16-bit), [signed] short [int] * (16-bit), [signed] [long] int, double, double *,……
- Other type, eg., bool, char are not directly supported.
- If Excel can’t convert input value, it will **not** call the function but will instead return **#VALUE!** error.

xl_array

- xl_array structure’s advantage
  - -Memory management is easy
  - -Accessing the data is simple
- xl_array structure’s disadvantage
  - -xl_array can only contain numbers
  - -Difficulties with the freeing of dynamic allocated memory. (Using static)
  - -This data type can’t be used for optional arguments

xloper--structure

- The xloper can represent
  - -Cell value
  - -Arrays
  - -Single cells, single block, multiple cells…
- The xloper structure contains two part :
  - -A 2-byte WORD – data type of xloper
  - -An 8-byte C union interpreted accroding to the type of xloper

xloper--structure (Cont.)

```
typedef struct xloper
{
    union
    {
        double num;                     /* xltypeNum */
        LPSTR str;                      /* xltypeStr */
        WORD bool;                      /* xltypeBool */
        WORD err;                       /* xltypeErr */
        short int w;                    /* xltypeInt */
        struct
        {
            WORD count;                 /* always = 1 */
            XLREF ref;
        } sref;                         /* xltypeSRef */
        struct
        {
            XLMREF far *lpmref;
            DWORD idSheet;
        } mref;                         /* xltypeRef */
        struct
        {
            struct xloper far *lparray;
            WORD rows;
            WORD columns;
        } array;                        /* xltypeMulti */
        struct
        {
            union
            {
                short int level;        /* xlflowRestart */
                short int tbctrl;       /* xlflowPause */
                DWORD idSheet;          /* xlflowGoto */
            } valflow;
            WORD rw;                    /* xlflowGoto */
            BYTE col;                   /* xlflowGoto */
            BYTE xlflow;
        } flow;                         /* xltypeFlow */
        struct
        {
            union
            {
                BYTE far *lpbData;      /* data passed to XL */
                HANDLE hdata;           /* data returned from XL */
            } h;
            long cbData;
        } bigdata;                      /* xltypeBigData */
    } val;
    WORD xltype;
} XLOPER, FAR *LPXLOPER;
```

xloper--structure (Cont.)

- xltype types table

```
int __stdcall xloper_type(xloper *p_op)
{
    if(p_op->xltype & xltypeStr)
        return xltypeStr;
    return 0;
}
```

Constant as defined in xlcall.h	Hexadecimal representation
xltypeNum	0x0001
xltypeStr	0x0002
xltypeBool	0x0004
xltypeRef	0x0008
xltypeErr	0x0010
xltypeMulti	0x0040
xltypeMissing	0x0080
xltypeNil	0x0100
xltypeSref	0x0400
xltypeInt	0x0800
xltypeBigData	0x0802