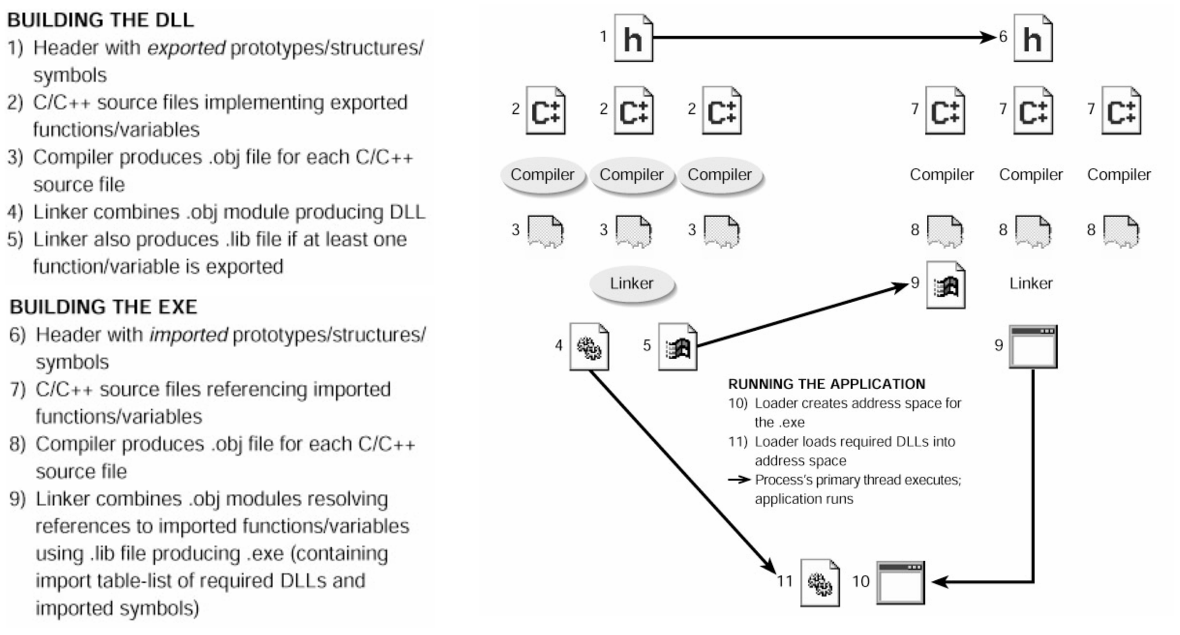
About XLL

DLL



BUILDING THE DLL

1) Header with *exported* prototypes/structures/symbols
2) C/C++ source files implementing exported functions/variables
3) Compiler produces .obj file for each C/C++ source file
4) Linker combines .obj module producing DLL
5) Linker also produces .lib file if at least one function/variable is exported

BUILDING THE EXE

6) Header with *imported* prototypes/structures/symbols
7) C/C++ source files referencing imported functions/variables
8) Compiler produces .obj file for each C/C++ source file
9) Linker combines .obj modules resolving references to imported functions/variables using .lib file producing .exe (containing import table-list of required DLLs and imported symbols)

RUNNING THE APPLICATION

10) Loader creates address space for the .exe
11) Loader loads required DLLs into address space

→ Process's primary thread executes; application runs

Functions to Export

- header file (.h)
  - __declspec(dllexport)

```
__declspec(dllexport)
     short __stdcall IsBusinessDay
     (long date, const char *holiday_centers);
```

- Or, exports file (.def)
  - EXPORTS

```
EXPORTS
     IsBusinessDay
```

- Processing
  - compiler will embed information into .obj
  - linker will produce a .lib containing symbols exported by DLL

```
dumpbin /exports example.lib
```

  - linker will also embed a table of exported symbols in DLL and the relative virtual address of each symbol

```
dumpbin /exports example.xll
```

__stdcall vs __cdecl

```
__declspec(dllexport)
     short __stdcall IsBusinessDay
     (long date, const char *holiday_centers);
```

- cleaning up of stack
  - caller for __cdecl (default)
    - Supporting variable number of parameters
  - callee for __stdcall

More on __stdcall

- Microsoft C compiler mangles C functions names

```
__declspec(dllexport)
       int __stdcall Add(int a, int b);

_Add@8
```

- Tell compiler not to mangle
  - linker directive

    ```
    #pragma comment(linker, "/export:Add=_Add@8")
    ```
  - Or, use .def

    ```
    EXPORTS
         Add
    ```

XLL

- XLL is simply a DLL that supports an interface through which Excel and DLL can communicate
  - DLL must export a number of functions for Excel to call
    - Initialize memory and data structures
    - Register function
    - Cooperate to manage memory
  - DLL needs access to functions to call Excel
    - Link to xlcall32.lib or xlcall32.dll
    - Call-back functions Excel4(), Excel4v(), XLCallVer()
    - Excel12(), Excel12v() (Excel 2007 v12)
- XLL loaded into Excel
  - *File/Open…*
  - *Tools/Add-ins…*

XLL Interface

- xlAutoOpen: Called when the XLL is loaded. The ideal place to register XLL functions and commands, initialize data structures, and customize the user interface.
- xlAutoClose: Called when the XLL is unloaded. The place to unregister functions and commands, release resources, and undo customizations.
- xlAutoAdd: Called when the XLL is activated (already loaded) or loaded during a session.
- xlAutoRemove: Called when the XLL is inactivated (without unloading) or unloaded during a session.

XLL Interface

- xlAddInManagerInfo (xlAddInManagerInfo12): Called when the add-in manager is invoked for the first time in this Excel session. If passed an argument = 1, it returns a string (the name of the add-in), otherwise it should return #VALUE!
- xlAutoRegister (xlAutoRegister12): Called when REGISTER (XLM) or xlfRegister (C API) is called without the function's return and argument types. It searches the XLL internally to register the function with this information supplied.
- xlAutoFree (xlAutoFree12): Called when Excel is returned an XLOPER flagged as pointing to memory that the XLL needs to release.

C API

- DLL/XLL to call
  - Excel Worksheet Functions
    - E.g. xlfLookup, xlfRound, xlfStdev, …
  - Macro Sheet Functions or Commands
    - E.g. xlfRegister, xlfUnregister, xlfCaller, xlfEvaluate, …
  - XLL-only Special Functions or Commands
    - E.g. xlFree, xlCoerce, xlSheetId, …
- via Callback Functions
  - Excel4(), Excel4v(), XLCallVer() - using XLOPER
    - exported by xlcall32.lib or xlcall32.dll
  - Excel12(), Excel12v() (Excel 2007 v12) - using XLOPER12
    - SDK C++ source file xlcall.cpp

xlfRegister

Argument	**Description**
1 (Req)	The full drive, path and filename of the DLL containing the function.
2 (Req)	The function name as it is exported. Note: This is case-sensitive.
3 (Req)	The return type, argument type and calling permission string.
4 (Req)	The function name as you wish it to appear in the worksheet. Note: this is case-sensitive.
5 (Req)	The argument names as a comma-delimited concatenated string, e.g., “Arg1, Arg2, Arg3”.

xlfRegister

Argument	Description
6 (Opt)	The function type: 1 or omitted = Function; 2 = Command.
7 (Opt)	The Paste Function category. If omitted the function is listed under "User Defined"
8 (Opt)	(Not used).
9 (Opt)	The help topic.
10 (Opt)	A brief description of the function, to be displayed in the Paste Function dialog.

xlfRegister

Argument	Description
11 (Opt)	Help for the 1st argument, to be displayed in the Paste Function dialog.
12 (Opt)	Help for the 2nd argument.
…	…
30 (Opt)	Help for the 20th argument

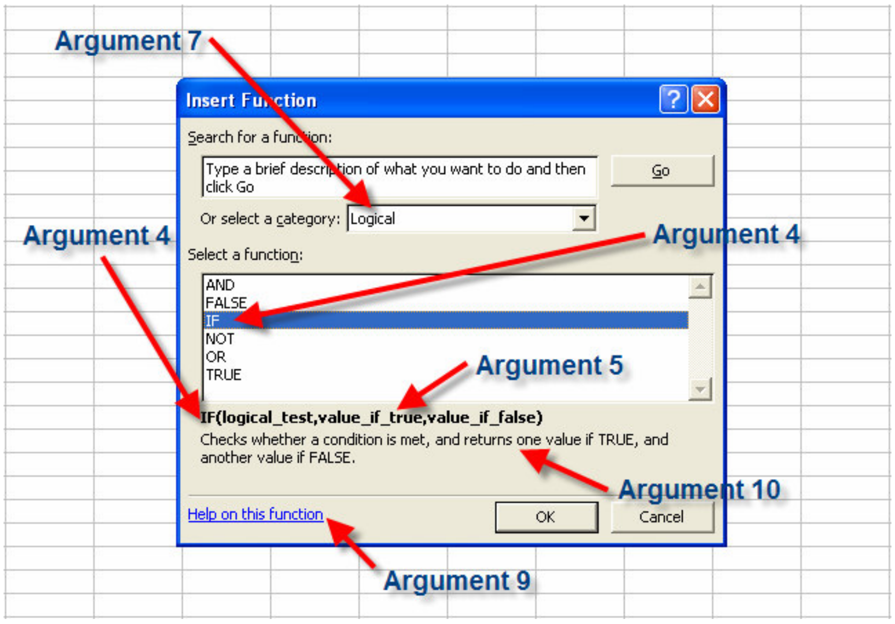
Standard worksheet function categories

Number	Text
1	Financial
2	Date & Time
3	Math & Trig
4	Text
5	Logical

Standard worksheet function categories

Number	Text
6	Lookup & Reference
7	Database
8	Statistical
9	Information
14	User Defined

Insert Function dialog



Search for a function:

Type a brief description of what you want to do and then click Go

Or select a category: Logical

Select a function:

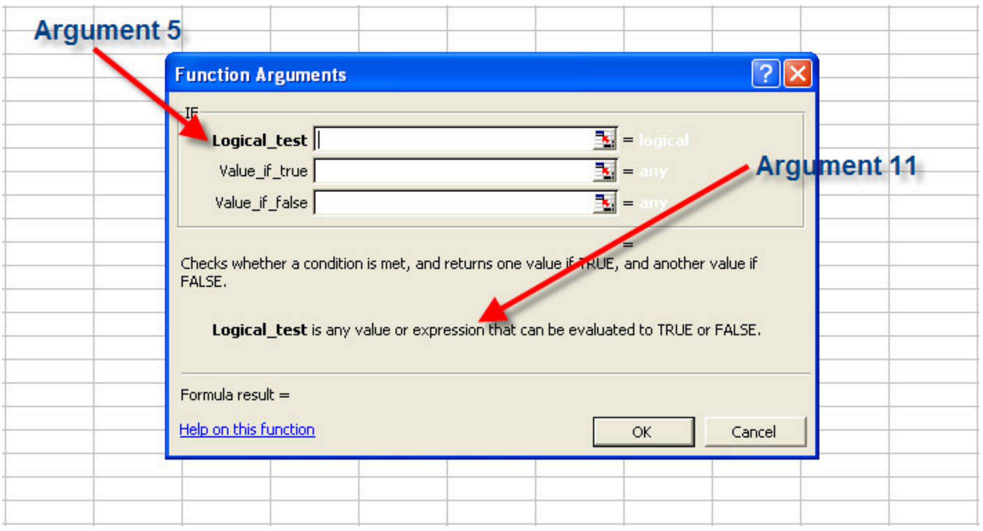
AND
FALSE
IF
NOT
OR
TRUE

IF(logical_test,value_if_true,value_if_false)

Checks whether a condition is met, and returns one value if TRUE, and another value if FALSE.

Help on this function

Insert Function dialog



IF

Logical_test = logical

Value_if_true = any

Value_if_false = any

Checks whether a condition is met, and returns one value if TRUE, and another value if FALSE.

Logical_test is any value or expression that can be evaluated to TRUE or FALSE.

Formula result =

Help on this function

How Excel Exchanges Worksheet Data with XLL

- Native C/C++ data types
- 2-D Array of 8-byte doubles (xl4_array)
  - xl_array.h
- xloper
  - Xlcall.h
  - xloper.xls

Registered function argument and return types

Data type	Pass by value	Pass by ref	Comment
Boolean	A	L	short (0 = false or 1 = true)
double	B	E	
char *		C F	Null-terminated string Modified-in-place
Unsigned char *		D G	Byte-counted string Modified-in-place
unsigned short [int]	H		DWORD, size_t, wchar_t (16-bit)

Registered function argument and return types

Data type	**Pass by value**	**Pass by ref**	**Comment**
[v12+] Unsigned short *		C% F%	Null terminated Unicode Modified-in-place
[v12+] unsigned short *	H	D% G%	Counted Unicode Modified-in-place
[signed] short [int]	I	M	16-bit
[signed long] int	J	N	32-bit

Registered function argument and return types

Data type	**Pass by ref**	**Comment**
FP (xl4_array)	K	2-D array of 8-byte doubles
[v12+] FP (xl12_array)	K%	Large 2-D array of 8-byte doubles
FP (xl4_array)	K	2-D array of 8-byte doubles
xloper [v12+]xloper12 xloper [v12+]xloper12	P Q R U	Variable-type worksheet values and arrays Values, arrays and range references

3 Class of Functions

- Excel4 and Excel12 distinguish between three classes of functions. The functions are classified according to the three states in which Excel might be calling the DLL.
  - Class 1 applies when the DLL is called from a worksheet as a result of recalculation.
  - Class 2 applies when the DLL is called from within a function macro or from a worksheet where it was registered with a number sign (**#**) in the type text.
  - Class 3 applies when a DLL is called from an object, macro, menu, toolbar, shortcut key, ExecuteExcel4Macro, or the Tools/Macro/Run command.

Permissions

- Class 1
  - Any worksheet function
  - Any XLL-only **xl**... function except **xlSet**.
  - **xlfCaller**
- Class 2
  - Any worksheet function
  - Any **xl...** function except **xlSet**.
  - Macro sheet functions, including **xlfCaller**, that return a value but perform no action that affects the workspace or any open workbook.
- Class 3
  - Any function, including **xlSet** and command-equivalent functions.

Volatile

- By default, DLL worksheet functions are not volatile
  - Only recalculate when their precedents change
  - Exception: function registered as # and type R (U)
- To make a DLL function volatile
  - Add an '!' at the end of the type string
  - Recalculate every time Excel performs a recalculation

Modifying in Place

- Argument is passed to DLL function via a pointer, DLL to return its value via this argument
  - In the 3rd argument of xlfRegister, instead of specifying a return type as the first character, a single digit from 1 to 9 informs Excel that the corresponding argument is to be used.
  - Function should declare void as return type
  - Excel will allocate the memory and clean up

Example Argument Strings (KJJ) - xl4_array

xl_array.cpp

```
{
        "xl_array_example1",
        "KJJ",
        "XlArrayExample1",
        "Rows,Columns",
        "1",
        "Example",
        "",
        "",
        "Returns array using FP structure",
        "Rows",
        "Columns",
        "",
},
```

Example Argument Strings (1K) - modified in place

xl_array.cpp

```
{
        "xl_array_example2",
        "1K",
        "XlArrayExample2",
        "InputArray",
        "1",
        "Example",
        "",
        "",
        "Returns array using FP structure modified in place",
        "InputArray",
        "",
        }
```

Example Argument Strings (RRJ#!) - permission

Excel4_examples.cpp

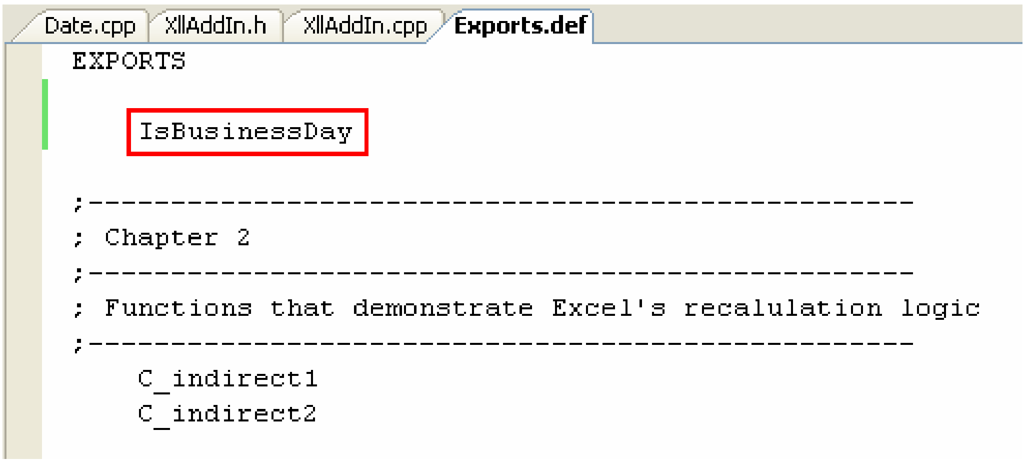
```
{
            "get_cell",
            "RRJ#!",
            "GetCell",
            "CellRef,InfoParam",
            "1",
            "Example",
            "",
            "",
            "Returns info about the given cell",
            "The cell ref in question",
            "An integer corresponding the info required",
            "",
},
```

Example Argument Strings (RJJ#!) - volatile

Excel4_examples.cpp

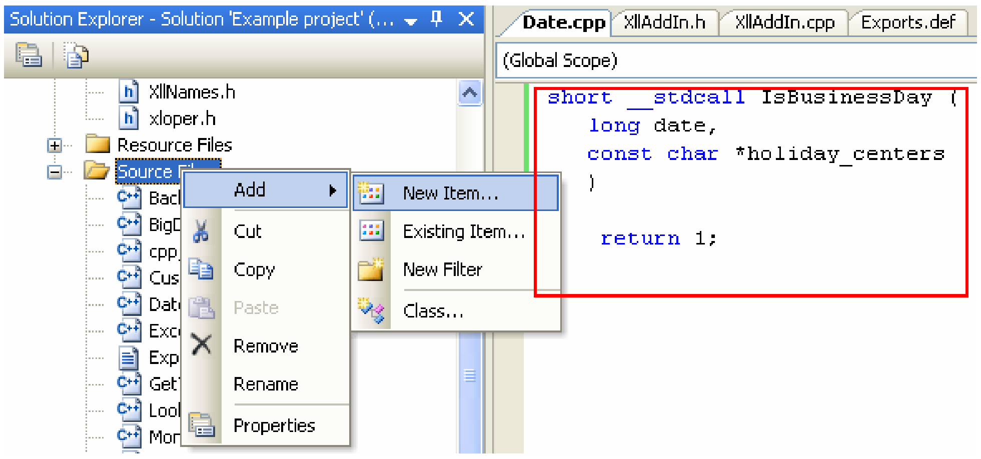
```
{
            "random_array",
            "RJJ#!",
            "RandomArray",
            "Rows,Cols",
            "1",
            "Example",
            "",
            "",
            "Returns array filled with random numbers",
            "Rows",
            "Cols",
            "",
},
```

Exports.def


Date.cpp | XllAddIn.h | XllAddIn.cpp | **Exports.def**

```
EXPORTS

    IsBusinessDay

;--------------------------------------------------
; Chapter 2
;--------------------------------------------------
; Functions that demonstrate Excel's recalulation logic
;--------------------------------------------------
    C_indirect1
    C_indirect2
```

Date.cpp

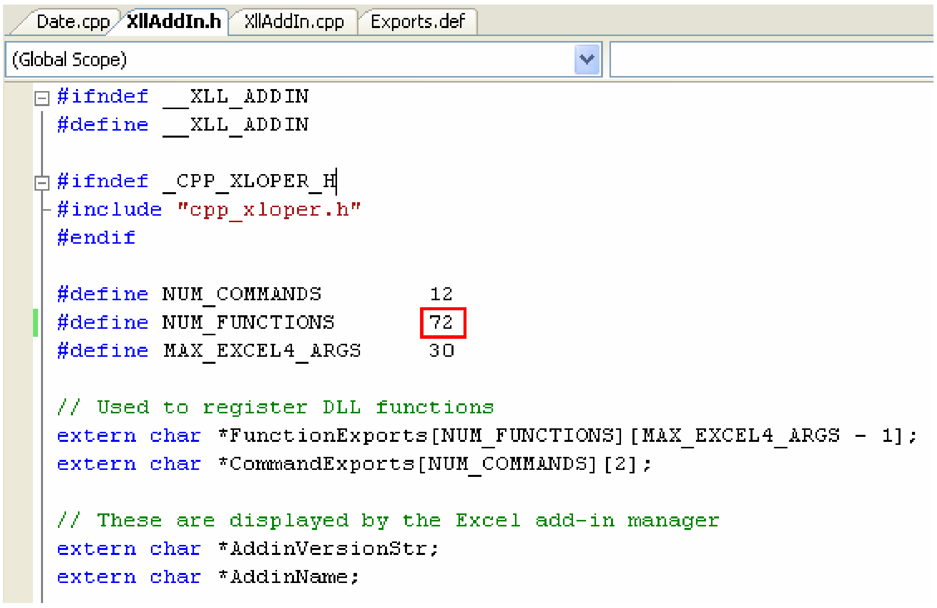


Date.cpp | XllAddIn.h | XllAddIn.cpp | Exports.def

(Global Scope)

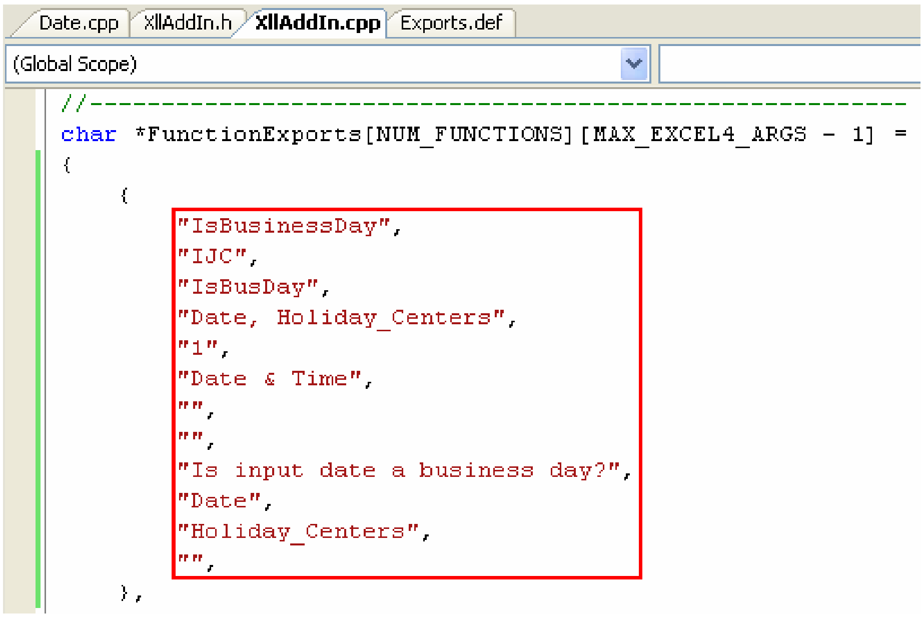
```
short __stdcall IsBusinessDay (
    long date,
    const char *holiday_centers
    )

    return 1;
```

XllAddIn.h



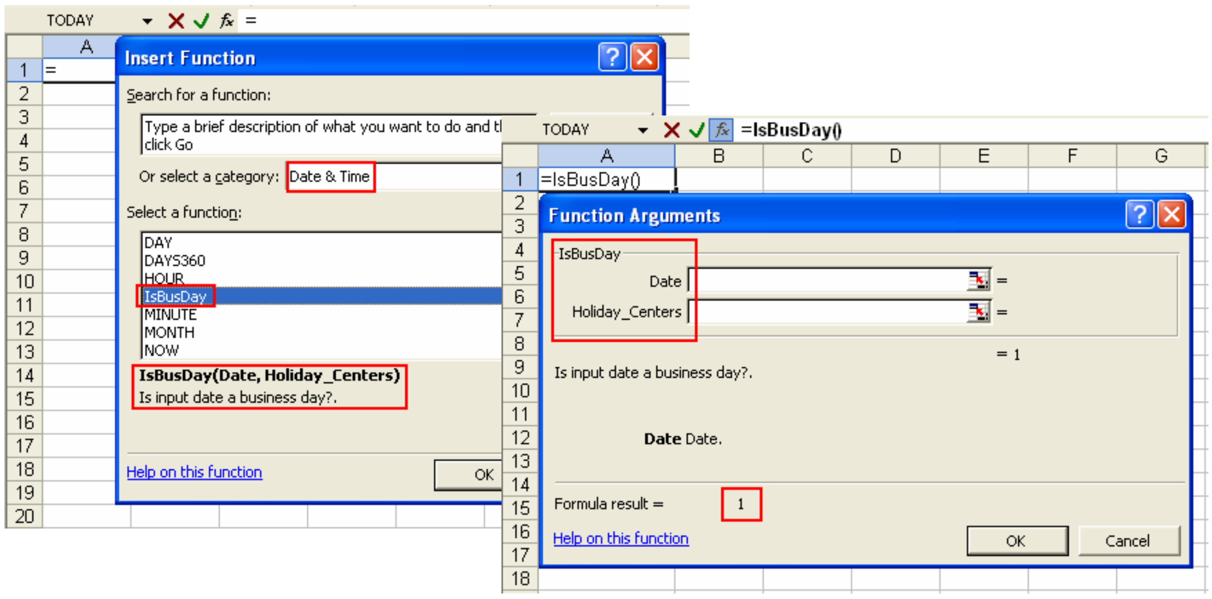
```
#ifndef __XLL_ADDIN
#define __XLL_ADDIN

#ifndef _CPP_XLOPER_H
#include "cpp_xloper.h"
#endif

#define NUM_COMMANDS        12
#define NUM_FUNCTIONS       72
#define MAX_EXCEL4_ARGS     30

// Used to register DLL functions
extern char *FunctionExports[NUM_FUNCTIONS][MAX_EXCEL4_ARGS - 1];
extern char *CommandExports[NUM_COMMANDS][2];

// These are displayed by the Excel add-in manager
extern char *AddinVersionStr;
extern char *AddinName;
```

XllAddIn.cpp



```
//----------------------------------------------------------------
char *FunctionExports[NUM_FUNCTIONS][MAX_EXCEL4_ARGS - 1] =
{
    {
        "IsBusinessDay",
        "IJC",
        "IsBusDay",
        "Date, Holiday_Centers",
        "1",
        "Date & Time",
        "",
        "",
        "Is input date a business day?",
        "Date",
        "Holiday_Centers",
        "",
    },
```

Insert Function



IsBusDay(Date, Holiday_Centers)
Is input date a business day?.

IsBusDay
Date =
Holiday_Centers =
= 1
Is input date a business day?.

Date Date.

Formula result = 1

Memory Management

- Memory considerations for the following three data structure types:
  - XLOPERs and XLOPER12s
  - Strings that are not in an XLOPER or XLOPER12
  - FP and FP12 arrays

XLOPER/XLOPER12

- An XLOPER/XLOPER12 can be created in several ways:
  - By Excel when preparing arguments to be passed to an XLL function
  - By Excel when returning an XLOPER or XLOPER12 in a C API call
  - By your DLL when creating arguments to be passed to a C API call
  - By your DLL when creating an XLL function return value
- A block of memory within one of the memory-pointing types can be allocated in several ways:
  - a static block within your DLL outside/within any function code, in which case you do not have to allocate or free the memory.
  - dynamically allocated and freed by your DLL in several possible ways: malloc and free, new and delete, and so on.
  - dynamically allocated by Excel.

Guidelines for XLOPER/XLOPER12

- Do not try to free memory or overwrite XLOPERs/XLOPER12s that are passed as arguments to your XLL function.
- You must only call xlFree on an XLOPER/XLOPER12 that was created as the return value to a C API call.
  - Example: getDllName() in Date.cpp
- If your DLL has allocated memory for an XLOPER/XLOPER12 that you want to return to Excel as your DLL function’s return value, you must tell Excel that there is memory that the DLL must release.
  - Example: getDllName() in Date.cpp
- If Excel has returned xltypeMulti to your DLL, do not overwrite any XLOPER/XLOPER12s within the array, especially if they contain strings, and especially not where you are trying to overwrite with a string.

Returning Modify-in-Place Arguments

- Only for argument passed in as a pointer
- Especially useful for
  - Length-counted and null-terminated ASCII byte strings
  - Length-counted and null-terminated Unicode wide character strings (Excel 2007 only)
  - FP floating-point arrays
  - FP12 floating-point arrays (Excel 2007 only)
- Should not return XLOPERs or XLOPER12s
- Excel allocates the memory for the return values
- Once Excel has finished reading the returned data, it releases the memory

String Types Supported C API xltypeStr XLOPER/XLOPER12s

Byte strings: XLOPER	**Wide character strings: XLOPER12**
All versions of Excel	Excel 2007+ only
Max length: 255 extended ASCII bytes	Maximum length: 32,767 Unicode chars
First (unsigned) byte = length	First Unicode character = length

Do not assume null termination of XLOPER or XLOPER12 strings.

String Types Supported C/C++ strings

Byte strings	**Wide character strings**
Null-terminated (char *) "C" Max length: 255 extended ASCII bytes	Null-terminated (wchar_t *) "C%" Maximum length: 32,767 Unicode chars
Length-counted (unsigned char *) "D"	Length-counted (wchar_t *) "D%"

Strings in xltypeMulti XLOPER/XLOPER12 Arrays

- Where encounter in DLL/XLL
  - Some C API functions return an xltypeMulti
  - Passed as an argument to an XLL function
  - Coerced to xltypeMulti from a range reference
- Excel creates deep copies of the strings in the source cells and points to these within the array
- To modify these strings in DLL/XLL, make own deep copies

Excel Stack

- Excel shares its stack space with all the DLL/XLLs it has loaded
- Guidelines
  - Do not pass very large structures as arguments to functions by value on the stack. Pass pointers or references instead.
  - Do not return large structures on the stack. Return pointers to static or dynamically allocated memory, or use arguments passed by reference.
  - Do not declare very large automatic variable structures in the function code. If you need them, declare them as static.
  - Do not call functions recursively unless you are sure the depth of recursion will always be shallow. Try using a loop instead.