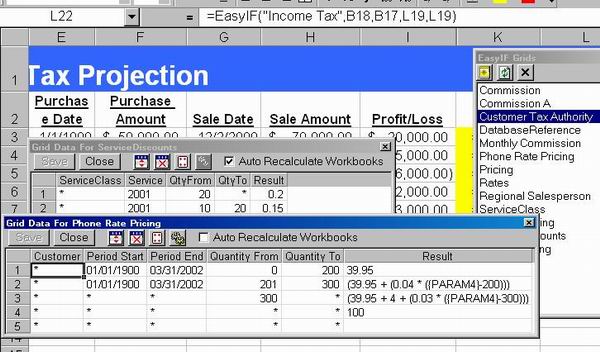
**Excel Add-in Using Excel C API and COM**

**By**[**basementman**](http://www.codeproject.com/script/Articles/MemberArticles.aspx?amid=68519)  
  
Demonstrates an Excel add-in that uses the excel4 C API as well as COM automation techniques.



**Introduction**

This sample add-in demonstrates how the Excel C API can be used to develop add-ins, in conjunction with COM automation. Back in the dark ages (before COM add-ins), programs like Excel and Word exposed a C API that developers could use to develop add-ins for. They even came up with special DLL file name extensions (XLL for Excel and WLL for Word) to denote these as Office add-in modules.

The general overview of how this application is constructed:

* An Excel C API based DLL module that interfaces with Excel
* Visual components added to the Excel menu structure using the C API
* Standard MFC-style coding for the add-in UI, dialogs, etc.
* COM automation to control certain aspects of Excel that are not exposed via the C API

The basic interface to the Excel C API involves exporting certain named functions that Excel calls automatically (similar to ISAPI). These functions then register the necessary Formulas, Menu items and Toolbars by interacting with Excel via the Excel4 function, passing the names and corresponding function pointers. Data is passed back and forth in this function using an unusual data structure, the XLOPER. This structure predates OLE and is very similar in concept to theVARIANT data type structure. There are some structure members and concepts specific to Excel, however. When the add-in functions are called, care must be taken not to assume that the function is running on any specific thread, since Excel calculation engine may call your function as part of a recalc, or the Excel UI may call the function as part of the formula builder.

**Overview Of The Addin's Functionality**

With EasyIf, simple to comprehensive logical functions such as if/then relationships, and lookup and reference functions such as VLOOKUP and HLOOKUP can be combined into one easy worksheet function, replacing complex and at times, limiting standard functions. This Excel add-in product is easy to use and maintain, and delivers powerful logical relationship comparisons, unprecedented lookup capabilities and accurate value selection that were only previously available through custom macro programming. The add-in contains a new comparative analysis function that, depending on the comparison criteria, Excel either does not offer or is so difficult and limiting to use via built-in logical and lookup/reference functions that users shun away from this comparative analysis thereby missing some of the value that Excel can provide. Click [here](http://test.pubsuite.com/jeff/EasyIFUserGuide.doc) for the official User's Guide for a more complete description of the functionality.

**Secondary Topic Of Interest**

The heart of the add-in's functionality involves a weighted pattern-matching engine. This engine allows for complex, exception based selection and determination. Consider the following example:

*The rate for Programming services is $100, except if it is done by a Senior developer then the rate is $120. But, if the work is done by someone from the San Francisco area, the rates are $115 and $135 respectively. These rates apply for all customers, except for GOLD level customers who receive a 10% discount. Oh, by the way, we have also negotiated specific, lower rates for special customers based on contracts.*

How would you:

1. code this so that it is maintainable and
2. be able to do an Excel spreadsheet on this?

With the Exception-based pattern-matching engine, you could set up a table that contains the following entries, where '\*' denotes a wildcard.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Cust Level** | **Customer** | **Location** | **Service** | **WorkerClass** | **Rate** |
| \* | \* | \* | PROG | \* | 100 |
| \* | \* | \* | PROG | SENIOR | 120 |
| \* | \* | SF | PROG | \* | 115 |
| \* | \* | SF | PROG | SENIOR | 135 |
| GOLD | \* | \* | PROG | \* | 90 |
| GOLD | \* | \* | PROG | SENIOR | 108 |
| GOLD | \* | SF | PROG | \* | 103.50 |
| GOLD | \* | SF | PROG | SENIOR | 121.50 |
| \* | Microsoft | \* | PROG | \* | 76 |
| \* | Microsoft | \* | PROG | SENIOR | 87 |

Using the pattern matching algorithm, this is all that would need to be maintained. The algorithm automatically selects the most fully matched entry, returning the rate in this case. Since it supports different data types for the columns, date-sensitivity could be added without any recoding by simply inserting additional columns. Columns can be configured to use extended comparison operators as well, including <, <=, >, >= and <>. Any number of these tables may be defined, each having a different structure and return type. For more information, consult the demo or *Matrix.h* and *Matrix.cpp*.

**Important Program Note**

The source code provided is for viewing only and will NOT compile, as it uses a third-party commercial library for certain functions. For those of you who have a license for Dundas Ultimate Grid 97 for MFC, you may have a chance of compiling it.

**Conclusion**

Hopefully, this code will illustrate that the good old fashioned C API can coexist with current COM technology, allowing the developer to use each technology where appropriate. Additionally, it provides a (hopefully) unique concept on how to easily handle complex decision trees and lookups of constantly changing values.

Top of Form

**License**

This article has no explicit license attached to it but may contain usage terms in the article text or the download files themselves. If in doubt please contact the author via the discussion board below.

A list of licenses authors might use can be found [here](http://www.codeproject.com/info/Licenses.aspx)

**About the Author**

|  |  |
| --- | --- |
| [**basementman**](http://www.codeproject.com/script/Membership/Profiles.aspx?mid=68519)  http://www.codeproject.com/script/Membership/ProfileImages/%7bC8F83E9B-E1DE-4BBA-B808-B295E23CB856%7d.gif | Starting developing in 1980's using Basic, MSC 5.0, Btrieve and DBFs. Back when men were men and there were no wizards. Since then, I have developed payroll processing system, general ledger packages, billing systems, project accounting systems, order entry, a report writer, email systems, development frameworks, C++ based Web Application Server and others that have long been forgotten.  Professionally, I have been a developer, sr. developer, development lead, architect, chief architect, CTO, director of development, and consultant. I am also in charge of changing the bottled water when empty. |

Bottom of Form