Creating A MSN-like Stock Quotes Add-In For Excel 2007
Using User-Defined Functions And Ribbons

Jürgen Bäurle

August 2007

Parago Media GmbH & Co. KG

Introduction

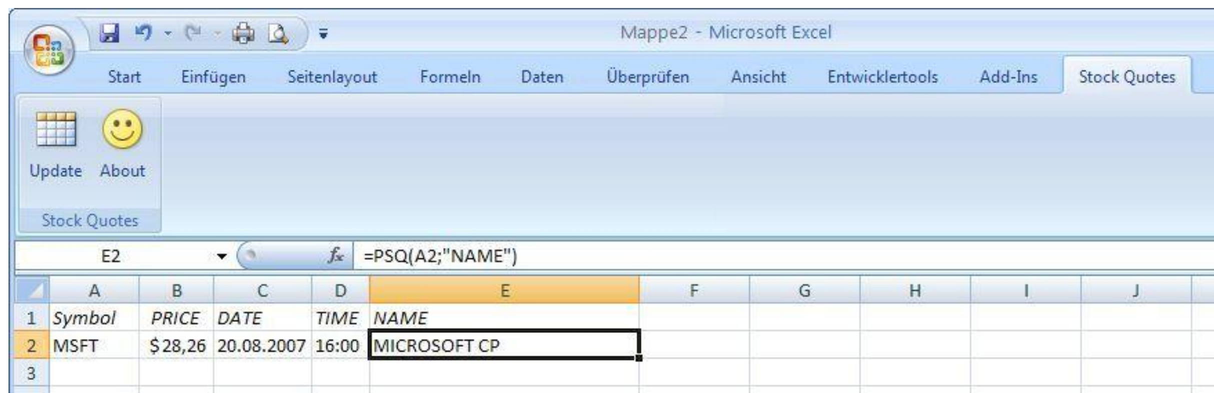
This article describes how to implement a managed stock quote add-in for Excel 2007 in C# that behaves similar to the "MSN MoneyCentral Stock Quotes" add-in you can download from Microsoft (see "*Excel 2003/2002 Add-in: MSN Money Stock Quotes*"). The add-in is developed from scratch, not using Visual Studio Tools for Office (VSTO).

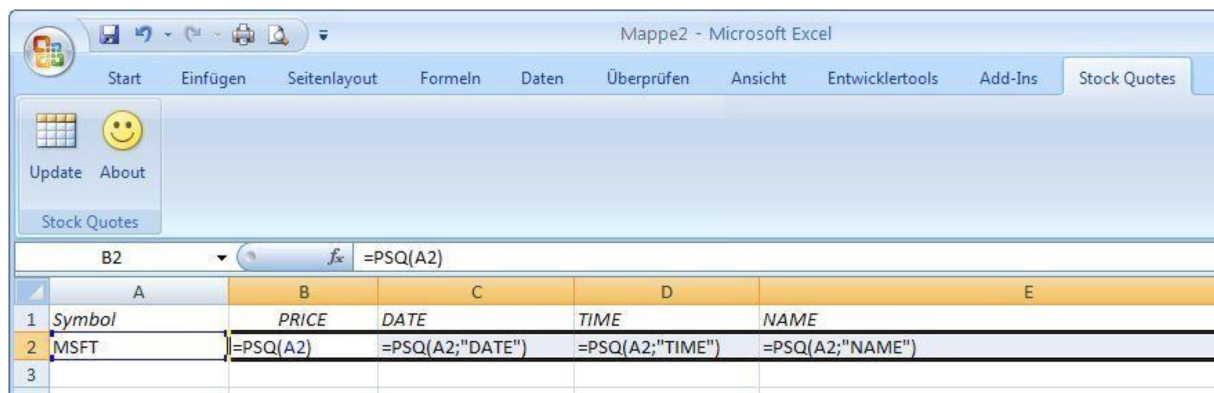
Once the add-in is installed and registered you can use the user-defined function (UDF) named "PSQ" to retrieve a stock quote price, date, time and name for a given stock symbol from the Yahoo finance service.

The following formula samples show the usage of the PSQ function:

> **`=PSQ(B4;'PRICE')`** or just **`=PSQ(A1)`** to retrieve the last price
> **`=PSQ(B4;'NAME')`** to retrieve the company name
> **`=PSQ(B4;'DATE')`** to retrieve the last price date
> **`=PSQ(B4;'TIME')`** to retrieve the last price time

The add-in is also implementing a so called custom ribbon of the new Office "Fluent" user interface. The following two screenshots shows the add-in in action using the German version of Excel:





The custom ribbon (named "Stock Quotes") offers an Update button to retrieve (to be more precise to recalculate) all stock related formulas at once.

The main objective of this composition is to show how easy you can create an Excel 2007 add-in using the new UI elements and especially user-defined functions. Therefore, to keep this task simple and straightforward, I omitted the setup project and the usually necessary COM shim project (see explanation below) for managed Office extensions.

Office Extensions

In the course of the years, Microsoft exposed a huge number of different Office extensibility mechanisms like COM add-ins, XLL extensions, smart tags and much more, that developers could use to create their own custom solutions.

The stock quotes add-in described in this article is a COM add-in, to be more exact, it is broken into two add-ins. One add-in is providing the user-defined function "PSQ", the other one is responsible for the UI elements (ribbon). This technique allows you to use the UDF without activating the UI.

A COM add-in is an ActiveX-DLL (in-process server), that implements the "*IDTExensibility*" interface and all COM add-ins must provide an implementation of those five interface methods: *OnAddInsUpdate*, *OnBeginShutdown*, *OnConnection*, *OnDisconnection*, *OnStartupComplete*. In our case we just provide empty methods with no specific logic, except of OnConnection. This override saves a given reference to the host application (Excel) for later use:

```
public void OnConnection(object host, ext_ConnectMode connectMode, object addInInst,
   ref Array custom)
{
   _excel = (Application)host;
}
```

In addition to the classic COM registration, the add-in must register with Excel or each other Office application the extension is working with. To register the extension with Excel, the add-in should create a sub-key, using its ProgID as the name for the key, under the following location:

*HKEY_CURRENT_USER\Software\Microsoft\Office\Excel\Addins**ProgID***

In our case the ProgIDs for the two COM add-ins are ***ParagoStockQuote.Functions*** and ***ParagoStockQuote.UI***. The add-in registration is providing more information values for Excel like a full description and a friendly name. In addition we specify the add-in LoadBehavior (0x03 for load on application start up).

In order to register the extension in the registry, the add-in provides two static methods named *RegisterFunction* and *UnregisterFunction*. They are attributed with *ComRegisterFunctionAttribute* and *ComUnregisterFunctionAttribute*. The CLR calls this methods once the containing assembly is registered through utilities like RegAsm:

```
[ComRegisterFunctionAttribute]
public static void RegisterFunction(Type t)
{
   RegistryKey key;

   key = Registry.ClassesRoot.CreateSubKey(Functions.ClsIdKeyName + "Programmable");
   key.Close();
   key = Registry.ClassesRoot.CreateSubKey(Functions.ClsIdKeyName + "InprocServer32");
   key.SetValue(string.Empty, Environment.SystemDirectory + @"\mscoree.dll");
   key.Close();

   key = Registry.CurrentUser.CreateSubKey(Functions.ExcelAddInKeyName);
   key.SetValue("Description", "Parago.de Stock Quotes Function Add-In for Excel 2007",
      RegistryValueKind.String);
   key.SetValue("FriendlyName", "Parago.de Stock Quotes Function Add-In",
      RegistryValueKind.String);
   key.SetValue("LoadBehavior", 3, RegistryValueKind.DWord);
   key.SetValue("CommandLineSafe", 1, RegistryValueKind.DWord);
   key.Close();
}

[ComUnregisterFunctionAttribute]
public static void UnregisterFunction(Type t)
{
   Registry.ClassesRoot.DeleteSubKey(Functions.ClsIdKeyName + "Programmable");
   Registry.ClassesRoot.DeleteSubKeyTree(Functions.ClsIdKeyName + "InprocServer32");

   Registry.CurrentUser.DeleteSubKey(Functions.ExcelAddInKeyName);
}
```

For more information about COM Interop and COM registration, I recommend you the CodeProject.com article "*Understanding Classic COM Interoperability With .NET Applications*".

Stock Quote Data

The add-in is calling the finance portal from Yahoo.com to retrieve the requested stock quotes. The following URL is returning a comma separated string containing the stock quote data:

http://download.finance.yahoo.com/d/quotes.csv?s=MSFT&f=sl1d1t1n

The query string parameters are "**s**" for the stock symbol and "**f**" for the returned stock data fields. You can find a list of available fields in the documentation of the Perl module "*Finance::YahooQuote*" (CPAN.org). So, using this source it's a matter of a few lines of code to get stock quotes (see code and section "User-Defined Functions").

User-Defined Functions

User-defined functions are not a big deal. They are just plain methods with or without optional parameters. Optional parameters are flagged with the *OptionalAttribute*. The method for the "PSQ" looks as follows:

```
public object PSQ(Range Cell, [Optional] object InfoCode)
{
   string symbol = Cell.Value2 as string;
   string infoCode = (InfoCode is Missing) ? "PRICE" : InfoCode as string;

   if(string.IsNullOrEmpty(symbol) || string.IsNullOrEmpty(infoCode))
      throw new Exception();

   WebClient client = new WebClient();
   Stream data = client.OpenRead("http://download.finance.yahoo.com/d/quotes.csv?s=" +
      symbol.Trim() + "&f=sl1d1t1n");
   StreamReader reader = new StreamReader(data);
   string content = reader.ReadToEnd();
   data.Close();
   reader.Close();

   string[] quote = content.Split(",".ToCharArray());

   switch(infoCode.Trim().ToUpper())
    {
      case "NAME":
         return quote[4].Replace("\"", "").Replace("\r", "").Replace("\n", "");
      case "DATE":
         return Convert.ToDateTime(quote[2].Trim("\"".ToCharArray()),
            CultureInfo.InvariantCulture).ToShortDateString();
      case "TIME":
         return Convert.ToDateTime(quote[3].Trim("\"".ToCharArray()),
            CultureInfo.InvariantCulture).ToShortTimeString();
      case "PRICE":
      default:
         return Convert.ToDouble(quote[1], CultureInfo.InvariantCulture);
   }
}
```

The UDF is expecting an Excel cell defined by a Range object and an optional parameter named InfoCode. The cell is referencing the stock symbol and InfoCode is defines the returning type like "NAME" or "DATE". If InfoCode is omitted (type of object is *System.Reflection.Missing*), then the stock price is returned. Throwing an exception is resulting in a "*#VALUE*" cell display.

In order to use the Range object and other Excel specific objects, a reference to the "*Microsoft.Office.Interop.Excel*" assembly must exist.

User Interface

The second add-in used in this project provides the user interface for the extension and consists of a ribbon element (see screenshot above). Apart from implementing the interface IDTExtensibility2, one must implement the interface *IRibbonExtensibility* to support ribbons in Excel or Office applications.

Extending Excel with a ribbon is basically just returning the XML definition of the ribbon element in the method *IRibbonExtensibility.GetCustomUI*. The add-in returns the ribbon definition as embedded resource. The user interface elements like buttons used by the ribbon are defined within the Ribbon.xml file:

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
   <ribbon startFromScratch="false">
      <tabs>
         <tab id="psqTab" label="Stock Quotes" insertAfterMso="TabAddIns">
            <group id="psqGroup" label="Stock Quotes">
               <button id="psqUpdateButton" label="Update" imageMso="TableStyleRowHeaders"
                  size="large" onAction="OnUpdateButtonClick" />
```

```
            <button id="psqAboutButton" label="About" imageMso="HappyFace" size="large"
               onAction="OnAboutButtonClick" />
          </group>
        </tab>
      </tabs>
    </ribbon>
</customUI>
```

The XML definition adds a new tab after (*insertAfterMso*) the "Add-Ins" tab and a group called "Stock Quotes" with two buttons. The button definitions are wiring up the click (*onAction*) events with event handlers. The click handler OnUpdateButtonClick is retrieving the stock quotes:

```
public void OnUpdateButtonClick(IRibbonControl control)
{
   _excel.CalculateFull();
}
```

The project must add a reference to the "*Microsoft.Office.Core*" assembly.

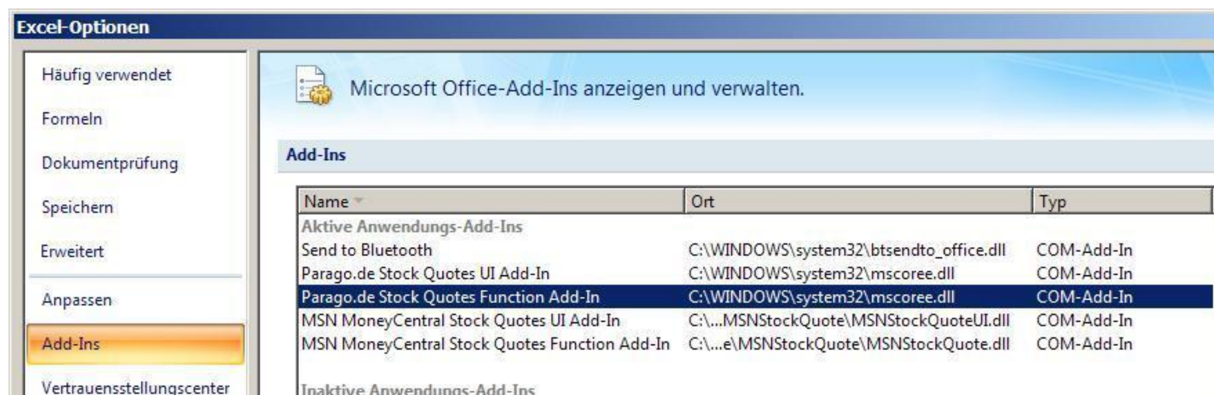
COM Shim

Even though this project does not contain a COM shim, I want to explain in short the idea behind this concept. A COM shim is basically an unmanaged wrapper (start module) of the managed extension to isolate the add-in in its own application domain. The main two reasons for using a custom COM shim (or the one provided by Visual Studio Tools for Office, called AddinLoader.dll) are security and isolation.

Office checks for digital signatures if the security level is set to high. If you deploy a managed add-in without the use of a COM shim, then Office loads the extension using mscoree.dll (.NET common language runtime engine). Since you cannot digitally sign the mscoree.dll, you need to create a COM shim, which can be signed and can be examined by Office for signatures.

If you do not use a any kind of COM shim, the add-in DLL loads into the default application domain along with all other un-shimmed extensions. All DLLs running in the same application domain are vulnerable to potential damage caused by any other DLL in the same application domain. Also, any un-shimmed add-in that crashes in a host application disables all other un-shimmed add-ins for that application

Also, the COM shim DLL with its location is showing up in the Excel Add-In dialog. If you do not use a unmanaged start module, the CLR engine mscoree.dll will be referenced, as the following figure shows:



A COM shim is not included with the sample code in this article. They are easily created using the "*COM Shim Wizard Version 2.3"* (see MDSN).

Summary

The add-in itself is a very basic implementation, but it shows how easy it could be to extend Excel using managed code. Creating an Office COM add-in is a straightforward task and you may use the "Shared Add-In" project wizard to create the boilerplate project code.

The add-in could be extended by creating a custom task pane to search stocks. The COM shim wizard (version 2.3) comes with a few managed samples, showing the use of task panes. The source code can be download from my homepage. For exact implementation details, please refer the source code. The project has been created in Visual Studio 2005.

Contact Information

If you have any feedback or suggestions, please feel free to contact me:

Jürgen Bäurle
jbaurle@parago.de
http://www.parago.de/jbaurle

Parago Media GmbH & Co. KG
Im Wengert 3 | 71336 Waiblingen, Germany | Phone +49.7146.861803 | Internet http://www.parago.de