**Theme**: F# in Dev10

# Overview

This walkthrough summarizes the experience using F# in Visual Studio 2010, on top of the .NET4.0 framework. F# is a fairly horizontal feature, aiming to enable for a range of usage styles. This walkthrough takes one path through the use of the F# language, libraries, editor, project system and debugger, but is of course necessarily incomplete in covering the full feature breadth that will be available for F# in Visual Studio 2010.

# Walkthrough

## Description

A financial modeler/developer wants to do historical analysis of stock price data. They want to start with some fairly informal analysis of the data, but hope to integrate the results into a full-blown analysis application developed by a colleague.

## Steps

#### Scripting with F#

1. **Create a new F# script**. Go to File | New | File, then select Script and F# Script. Save as “stockanalysis.fsx”.
2. **Use .NET APIs from F# to pull in data from the Web**. Type in the code below.
   * Notice: Colorization of strings and keywords
   * Notice: Completion lists appear after every “.” Is typed
   * Notice: Invoking “Display Word Completion” (ctrl-space, ctrl-j, ctrl-k,w) in the middle of an identifier brings up a completion list
   * Notice: Hovering over any identifier in the code provides QuickInfo
   * Notice: F1 on ‘WebRequest’ produces expected documentation
   * Notice: F1 on “let” produces expected documentation
   * Notice: Types and namespaces from at least mscorlib, System.dll and System.Windows.Forms.dll are referenced by default.

open System.Net

open System.IO

let url = "http://ichart.finance.yahoo.com/table.csv?s=MSFT&d=10&e=10&f=2008&g=d&a=2&b=13&c=1986&ignore=.csv"

let req = WebRequest.Create(url)

let resp = req.GetResponse()

let stream = resp.GetResponseStream()

let reader = new StreamReader(stream)

let csv = reader.ReadToEnd()

1. **Execute code with F# Interactive**. Select all the code (ctrl-A) and right-click, then select ‘Send to F# Interactive’ (alternatively, alt-enter).
   * Notice: If not visible already, the F# Interactive appears
   * Notice: Code executes successfully
   * Notice: F# Interactive responds with:

val url : string = <string:97>

val req : System.Net.WebRequest = System.Net.HttpWebRequest

val resp : System.Net.WebResponse = System.Net.HttpWebResponse

val stream : System.IO.Stream = System.Net.ConnectStream

val reader : System.IO.StreamReader = System.IO.StreamReader

val csv : string = <string:287846>

>

1. **Inspect data with the F# Interactive**. At the F# Interactive prompt, type “csv;;” then enter. Also type “csv.Length;;” then enter.
   * Notice: Date is “live”
   * Notice: The F# interactive prints the value of the string ‘csv’, and the length.

1986-03-18,29.50,29.75,28.50,28.75,67766400,0.08

1986-03-17,29.00,29.75,29.00,29.50,133171200,0.09

1986-03-14,28.00,29.50,28.00,29.00,308160000,0.08

1986-03-13,25.50,29.25,25.50,28.00,1031788800,0.08

"

> csv.Length;;

val it : int = 287846

>

1. **Write F# code to parse CSV data**. In the editor, add the code below. As each line is added, select the code added in this section so far, and hit alt-enter to see the partial results.
   * Notice: Continue to get colorization, quick info and correct completion lists on “.” even in the middle of complex nested expressions.
   * Notice: While code is incomplete (or incorrect), syntactic and semantic error squiggles appear in the code.
   * Notice: Pipelines and F# Interactive allow easy partial execution of data processing code.

let prices =

csv.Split([|'\n'|])

|> Seq.skip 1

|> Seq.map (fun line -> line.Split([|','|]))

|> Seq.filter (fun values -> values.Length = 7)

|> Seq.map ( fun values ->

System.DateTime.Parse(values.[0]),

float values.[6])

1. **Give this functionality a name**. Remove MSFT from definition of ‘url’, and replace with ”+ticker+”. Add let ticker = “MSFT” on the line above. Select all code except this one new line, and hit <tab>. Above the indented block of code, add let loadPrices ticker =. At the end of the indented block, add prices.
   * Notice: Indentation is significant in F# - indicates nesting level.
   * Notice: Lightweight code abstraction provided by the language syntax
   * Notice: <tab> is almost like the ‘Extract Method’ refactoring!
   * Notice: Code now shows the following:

open System.Net

open System.IO

let ticker = "MSFT"

let loadPrices ticker =

let url = "http://ichart.finance.yahoo.com/table.csv?s="+ticker+"&d=10&e=10&f=2008&g=d&a=2&b=13&c=1986&ignore=.csv"

let req = WebRequest.Create(url)

let resp = req.GetResponse()

let stream = resp.GetResponseStream()

let reader = new StreamReader(stream)

let csv = reader.ReadToEnd()

let prices =

csv.Split([|'\n'|])

|> Seq.skip 1

|> Seq.map (fun line -> line.Split([|','|]))

|> Seq.filter (fun values -> values.Length = 7)

|> Seq.map ( fun values ->

System.DateTime.Parse(values.[0]),

float values.[6])

prices

1. **Use this functionality on new inputs**. Select all the code and hit alt-enter to execute with F# Interactive. At the F# Interactive prompt, call the new ‘loadPrices’ function on other ticker symbols: ‘GOOG’, ‘AAPL’, ‘ORCL’, ’EBAY’.
   * Notice: Old definitions are not lost in F# Interactive, but new definitions are available as well.
   * Notice: Non-trivial structured data is rendered by the pretty printer.

> loadPrices "EBAY";;

val it : seq<System.DateTime \* float>

= seq

[(11/10/2008 12:00:00 AM {Date = ...;

<...snip...>

TimeOfDay = 00:00:00;

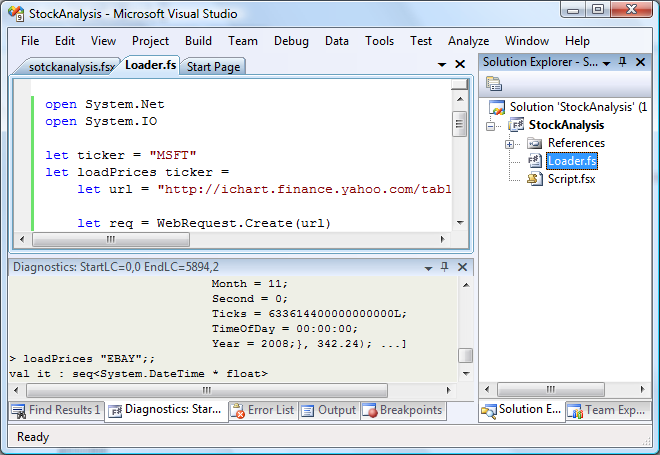
Year = 2008;}, 14.72); ...]

>

1. **Display in DataGrid and Plot using “F# for Visualization”**. COMING SOON TO WALKTHROUGH!

#### Component Development with F#

1. **Create a Library project to expose this functionality.**  File | New Project… | Visual F# | Library Project to create a new Library project. Copy/paste code from stokanalysis.fsx into Module1.fs. In Solution Explorer, rename “Module1.fs” to “StockLoader.fs”.
   * Notice: Visual F# has project templates
   * Notice: Visual F# is listed at top level in New Project Dialog if F# profile was selected.
   * Notice: Default F# Library template provides a code file (.fs) and a script (.fsx)



1. **Author a new F# class which exposes desired functionality**. In Solution Explorer, right click on the project node and select Add | New Item… then F# Source File. Name it “StockAnalysis.fs”. Select Script.fsx in the solution explorer and right-click then Move Down (alternatively alt-down). Type the code below in this file.
   * Notice: In Solution Explorer, file is added at the end, not in alphabetical order.
   * Notice: In Solution Explorer, context menu commands and alt-up, alt-down keybindings allow reordering files.
   * Notice: Can author OO .NET classes naturally in F#.

namespace StockAnalysis

open StockLoader

/// Provides analysis of historical stock data

type StockAnalyzer(lprices, days) =

let prices =

lprices

|> Seq.map snd

|> Seq.take days

/// Construct StockAnalyzer objects for each ticker over the

/// given number of days

static member GetAnalyzers(tickers, days) =

tickers

|> Seq.map loadPrices

|> Seq.map (fun lprices -> new StockAnalyzer(lprices, days))

member sa.Return =

let lastPrice = prices |> Seq.nth 0

let startPrice = prices |> Seq.nth (days - 1)

lastPrice / startPrice - 1.0

member sa.MeanAndStdDev =

let logRets =

prices

|> Seq.pairwise

|> Seq.map (fun (x,y) -> log (x/y))

let mean = logRets |> Seq.average

let sqr x = x \* x

let variance = logRets |> Seq.averageBy (fun x -> sqr (x - mean))

(mean, sqrt variance)

1. **Build**. Hit ctrl-shift-b or F6 to build.
   * Notice: Build succeeds
   * Notice: Error List shows no errors
   * Notice: Output window shows the following:

------ Build started: Project: StockAnalysis, Configuration: Debug Any CPU ------

c:\dd\fsharp\staging\Debug\\bin\fsc.exe -o obj\Debug\StockAnalysis.dll -g --noframework --define DEBUG --define TRACE -r "C:\Program Files\Reference Assemblies\Microsoft\Framework\v3.5\System.Core.dll" -r C:\Windows\Microsoft.NET\Framework\v2.0.50727\System.dll --target library --warn 3 --warnaserror 76 --vserrors --utf8output --fullpaths --flaterrors StockLoader.fs StockAnalysis.fs

StockAnalysis -> C:\prj\StockAnalysis\StockAnalysis\bin\Debug\StockAnalysis.dll

========== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ==========

1. **Add a C# client application**. Right click the solution and select Add | New Project… then C# then Visual C# | Console Application. Name it “CSharpDriver”. Right click it’s References node and select Add Reference… then in Projets tab, select StockAnalysis. Right click the project node and select Set as Startup Project. Type the code below in the body of the Main method.
   * Notice: Can add project-to-project references to/from C#/F#
   * Notice: F# defined namespaces and types can be used from C# like any other type
   * Notice: F# doc-comments are available in C# intellisense
   * Notice: C# can consume Tuple returns from F# API

var tickers = new[] { "MSFT", "GOOG", "ORCL", "EBAY" };

var analyzers = StockAnalysis.StockAnalyzer.GetAnalyzers(tickers, 365);

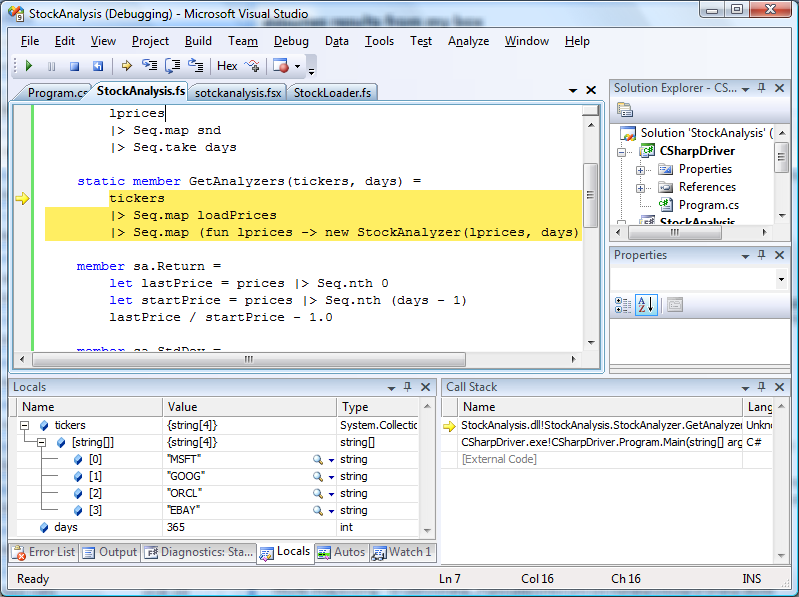
foreach (var item in analyzers)

{

Console.WriteLine("Return = {0}, \t Mean = {1}, \t StdDev = {2}", item.Return, item.MeanAndStdDev.Item0, item.MeanAndStdDev.Item1);

}

1. **Debug**. Hit F11 to build, launch under debugger, and step into. Hit F11 a few more times until stepping into F# code in the body of the GetAnalyzers member.
   * Notice: Debugging from C# into F# works seamlessly.
   * Notice: Debugging spans are “expression-based”.
   * Notice: Locals window shows the values of “tickers” and “days”
   * Notice: Continuing to step walks through the evaluation of the rest of the application
   * Notice: Debugger commands like “Run to cursor”, “Set next statement”, “Insert Breakpoint”, “Add Watch” and “Go to Disassembly” all work as expected.



1. **Historical Debugger**. COMING SOON!

#### Asynchronous Programming with F#

1. **F# PowerPack.** Go to live.com and search for the F# PowerPack. On the codeplex page, select the Downloads tab, and download the F# PowerPack to the machine. In the StockAnalysis project, righ click Add Reference… then browse to the FSharp.PowerPack.dll assembly and select it.
   * **Notice:** The F# powerpack is a set of shared source extensions to F#
   * **Notice:** The F# PowerPack can version independently of the core F# release
   * **Notice:** The F# PowerPack contains a few assemblies and tools
   * **Notice:** the F# PowerPack is set to Copy Local when refrenced
2. **Asynchronous Workflows**. In StockLoader.fs, make the changes on the four lines indicated below.
   * **Notice**: Asynchronous code maintains the same programming style as synchronous code.
   * **Notice**: “async” keyword introduces an asynchronous block of code, with “let!” indicating an asynchronous action inside the async code.

open System.Net

open System.IO

let ticker = "MSFT"

let loadPrices ticker = async {

let url = "http://ichart.finance.yahoo.com/table.csv?s="+ticker+"&d=10&e=10&f=2008&g=d&a=2&b=13&c=1986&ignore=.csv"

let req = WebRequest.Create(url)

let! resp = req.AsyncGetResponse()

let stream = resp.GetResponseStream()

let reader = new StreamReader(stream)

let! csv = reader.AsyncReadToEnd()

let prices =

csv.Split([|'\n'|])

|> Seq.skip 1

|> Seq.map (fun line -> line.Split([|','|]))

|> Seq.filter (fun values -> values.Length = 7)

|> Seq.map ( fun values ->

System.DateTime.Parse(values.[0]),

float values.[6])

return prices }

1. **Consuming Asynchronous Code**. Open the error list, and see that there is now an error in StockAnalysis.fs. Open the file, and change the GetAnalyzers as indicated below.
   * **Notice**: Error list works across files opened in the editor, and guides refactoring
   * **Notice**: Background type checking hints at the appropriate fixes below (how to turn a seq<async<’a>> into a seq<’a>?)
   * **Notice**: Async.Parallel provides a way to get parallelism using asynchronous workflows

/// Construct StockAnalyzer objects for each ticker over the

/// given number of days

static member GetAnalyzers(tickers, days) =

tickers

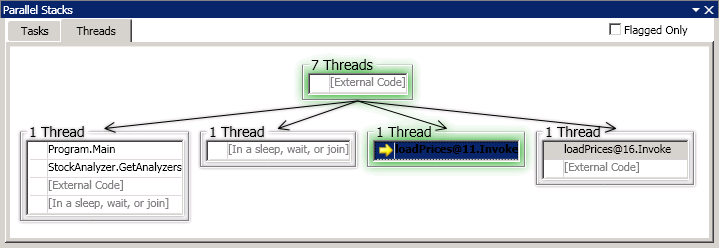
|> Seq.map loadPrices

|> Async.Parallel

|> Async.Run

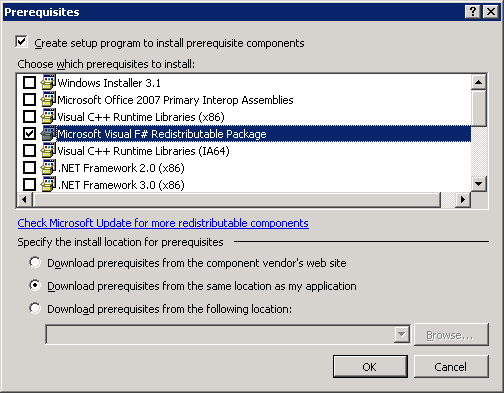
|> Seq.map (fun lprices -> new StockAnalyzer(lprices, days))

1. **Parallel Stacks Debugging Window**. Set a breakpoint on the line “let stream = …” in StockLoader.fs. Hit F5 to run to the breakpoint. In the main menu, select Debug->Windows->Parallel Stacks to open the window shown below. Press F5 a few (3-4) times.
   * **Notice:** You can see how the asynchronous workflow is executing on multiple threads in the Parallel Stacks window
   * **Notice:** Periodically, there will be two threads executing loadPrices, indicating CPU parallelism
   * **Notice:** Almost always there will be a thread in a sleep, wait or join, often indicating I/O parallelism.

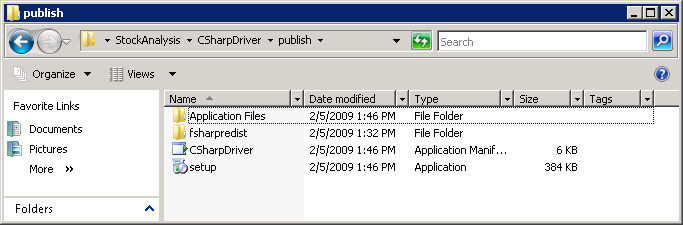


***Deploying the Application***

1. Select Prerequisites. Double-click the Properties node in CSharpDriver and open the Publish tab. Select the “Prerequisites…” button and check the box for “Microsoft Visual F# Redistributable Package”.
   * **Notice**: F# (currently) has a separate redist.
   * **Notice**: This redist must be explicitly added as a prerequisite.



1. **Deploy the C# application with ClickOnce**. Right click on the CSharpDriver project and select Publish… then Finish. Run the resulting CsharpDriver.exe.
   * Notice: F# Redist is included with the application
   * Notice: Running the application installs the F# redist and executes the app successfully.



## Features

1. F# Language
2. F# Libraries
3. F# Interactive
4. F# Language Service
5. F# Project System
6. Debugger integration
7. Mixed C#-F# development
8. Deployment