# Writing Valid F#

## Lab Overview

In this lab session, you will explore write a simple WebBrowser application in F#. You will gain a practical understanding of how to use values and functions as well as seeing how the F# type inference engine in action.

## Prerequisites

* Visual Studio 2015

## Time Estimate

* 30 minutes

## Table of Contents

[Writing Valid F# 1](#_Toc431682020)

[Lab Overview 1](#_Toc431682021)

[Prerequisites 1](#_Toc431682022)

[Time Estimate 1](#_Toc431682023)

[Table of Contents 1](#_Toc431682024)

[Exercise 1: Creating and using .NET objects from the F# 3](#_Toc431682025)

[Exercise 2: Controlling the Browser interactively 4](#_Toc431682035)

[Exercise 3: Experimenting with F# Type Inference 5](#_Toc431682041)

## Exercise 1: Creating and using .NET objects from the F#

In this exercise you will create a Web Browser within a Windows Form, download some HTML and display it in the Web Browser control, all through the F# REPL.

Create a new F# script in Visual Studio.

1. Create a let binding for a System.Net.WebClient.

Remember that types that implement IDisposable should be created with the **new** keyword.

1. Create a let binding for the contents of the fsharp.org website by calling the DownloadString() function on the WebClient to download the contents for <http://fsharp.org>. We will use this content later to display on the form.
2. You may find it easier at this point to open the System.Windows.Forms namespace.

Use the **open <namespace>** syntax to open a namespace e.g. **open System**. Make sure you execute this line into FSI as normal – there is no response in FSI when you do this aside from a blank line appearing.

1. Create a let binding for a System.Windows.Forms.Form
2. Set the following properties of the form: -

|  |  |
| --- | --- |
| Property | Value |
| Text | “Hello from F#” |
| Height | 600 |
| Width | 600 |

Remember that you can use two alternative ways of setting mutable properties on an object: -

form.Property <- value

form.OtherProperty <- otherValue

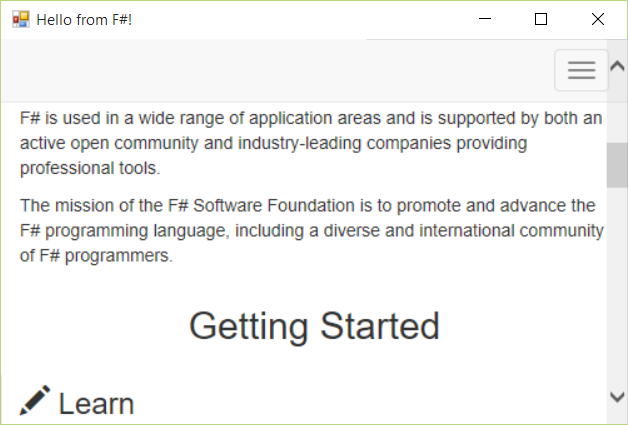
Alternatively, you can use constructor-time assignment.

Form(Property = value, OtherProperty = otherValue)

1. Create a System.Windows.Forms.WebBrowser control. Set its properties as follows: -

|  |  |
| --- | --- |
| Property | Value |
| ScriptErrorsSuppressed | true |
| Dock | DockStyle.Fill |
| DocumentText | <the value of the fsharp.org website let binding from Step 2> |

1. Add the WebBrowser control that you just created to the Form’s Controls collection.
2. Display the Form on screen by calling the Show() method on it. The following form should be shown.



1. As an extra exercise, refactor your code so that all let bindings are scoped correctly within multiline values. For example, you can create the WebClient within the creation of the value of fsharpOrg html content value, which in turn can be created within the creation of the WebBrowser object e.g.

let webClient = new WebClient()

let fsharpOrg = webClient.DownloadString("http://fsharp.org/")

can be rewritten as: -

let fsharpOrg =

let webClient = new WebClient()

webClient.DownloadString("http://fsharp.org/")

1. Turn your code into a function that takes in the URL to download. It will call Show() on the form that it creates, and then return the WebBrowser control to the caller. The function heading should look as follows: -

let showUrlInBrowser (url : string) : WebBrowser = …

i.e. the function showUrlInBrowser will take in a string and return a Form. As a side-effect, it will also show the Form.

**Working with IDisposable in F#**

You have already seen that you should use the new keyword to create Disposable objects. Does F# have an equivalent of the *using* keyword from e.g. C#? Yes! All you have to do is replace the *let* keyword with the *use* keyword. This will instruct F# to call *Dispose()* when the binding goes out of scope.

To prove this, replace the *let* bindings with *use* bindings for when you create the **WebClient**, **WebBrowser** and **Form** objects. What is the impact of doing this? Why?

## Exercise 2: Controlling the Browser interactively

In this exercise, we’ll enhance the code we wrote, in order to allow us to send new URLs to the web browser from FSI.

1. Create a new function which has a signature as follows: -

let updateBrowserContent(newUrl:string, browser:WebBrowser) : browser = …

1. The function should download the content of the url using a WebClient object, and then set the DocumentText property of the WebBrowser control, before returning the browser back out again.
2. Call the original showInBrowser function that you created earlier, and bind the resulting WebBrowser value to a symbol. You can now pass that object to the updateBrowserContent function with any URL and update the existing window with it e.g.

let browser = showUrlInBrowser ("http://fsharp.org")

updateBrowserContent(“<http://c4fsharp.net>”, browser)

updateBrowserContent(“http://fsprojects.github.io/”, browser)

updateBrowserContent(“https://sergeytihon.wordpress.com/category/f-weekly/”, browser)

## Exercise 3: Experimenting with F# Type Inference

In this exercise, we’ll explore how type inference works in F#.

1. Open the Type Inference.fsx file.
2. Experiment with changing the types of data returned from child functions e.g. Have calculateLightLevel() return 123 instead of “rainy day” or “sunny day”. What impact does this have on function type signatures?
3. Place explicit type annotations on functions or types. What impact does this have if you intentionally change the type signature to something that breaks the code?