Developing Dynamics CRM Plugins in F#

Microsoft Dynamics 365

Brown Bag Session

Martin Hornáček

martin.hornacek@accenture.com

What is F# ❤️

- **General-purpose** programming language.
- Mature, open source, cross-platform (**Windows**, **Mac** and **Linux**), functional-first programming language.
- **Strongly typed** language that uses **type inference**.
- Although a **functional language** at heart, F# does support other styles which are not 100% pure. This makes it easier to interact with the non-pure world of `web sites`, `databases`, `other applications`, and so on.

What is F# ❤️

- F# is designed as a **hybrid functional/OO language**, so it can do virtually `everything` that C# can do.
- **Functions** are first class objects, it is very easy to create powerful and reusable code by creating functions that have other functions as parameters, or that combine existing functions to create new functionality.
- F# is part of the **.NET ecosystem**, which gives you seamless access to all the third party .NET libraries and tools.

What is F# ❤️

- Runs on most platforms, including **Linux** and smart phones (via **Mono**).
- It is well integrated with **Visual Studio**, which means you get an IDE with IntelliSense support, a debugger, and many plug-ins for unit tests, source control, and other development tasks.

Why use F# ❤️

- Powerful type system prevents many common errors such as **null reference** exceptions.
- Values are **immutable** by default, which prevents a large class of errors.
- In addition, you can encode business logic using the type system itself in such a way that it is actually impossible to write incorrect code.

Why use F# ❤

- Lightweight syntax
- Immutable by default
- Type inference and automatic generalization
- Powerful data types

Why use F# ❤

- Pattern matching
- Supports exploratory style of programming
- Makes you focus on what now how
- Allows making illegal state unrepresentable

Why use F# ❤️

Demo

F# vs. C#

"Sum the squares from 1 to n" problem

```
public static class SumOfSquaresHelper
{
   public static int Square(int i)
   {
      return i * i;
   }

   public static int SumOfSquares(int n)
   {
      int sum = 0;
      for (int i = 1; i <= n; i++)
      {
         sum += Square(i);
      }
      return sum;
   }
}
```

F# vs. C#

"Sum the squares from 1 to n" problem

```
// define the square function
let square x = x * x

// define the sumOfSquares function
let sumOfSquares n =
   [1 .. n] |> List.map square |> List.sum

// try it
sumOfSquares 100
```

F# vs. C#

- F# can be developed interactively.
- F# can do anything C# can do.
- F# code is more compact.
- F# code didn’t have any type declarations.

Writing plug-ins in F# ❤️

- Standard development flow when developing plug-ins:
    - Develop and build your project.
    - Register the assembly.
    - Add your plugin steps.
    - Test.
- Actually trickier with **F#** than with **C#**.

Writing plug-ins in F# ❤

- You'll need **sign** the assembly first.
- The project properties UI doesn’t have a **Signing** tab, so you will have to create a key pair file manually.
- Use the **sn.exe** tool like this:

```
\sn.exe -k <KeyName>.snk
```

Writing plug-ins in F# ❤

- The plug-in needs reference to the **FSharp.Core.dll**.
- Without it there will be an **“Unexpected Error”** meesage shown in CRM.
- Possible solutions:
    - GAC
    - ILMerge
    - Server deployment

Writing plug-ins in F# ❤

Demo

Testing your F# code ❤️

- Support for all popular unit testing .NET frameworks:
  - MSTest, NUnit, xUnit
- Very suitable for **TDD** - small functions that give deterministically repeatable results lend themselves perfectly to unit tests.
- There are also **testing tools** written in or specifically for F#:
  - FsTest, FsUnit, FsCheck, Foq

F# Type Providers ❤️

- **FSharp.Data** includes type providers for JSON, XML, CSV and HTML document formats.
- **SQLProvider** provides strongly-typed access to SQL databases through a object mapping and F# LINQ queries against these data sources.
- **FSharp.Data.SqlClient** has a set of type providers for compile-time checked embedding of T-SQL in F#.
- **FSharp.Data.TypeProviders** is an older set of type providers for use only with .NET Framework programming for accessing SQL, Entity Framework, OData and WSDL data services.

F# Type Providers ❤

- The **DynamicsCRMProvider** is a `F# type provider`
- It allows to access CRM data in a strongly typed way.

```
// load the type provider dependencies
#load @"..\packages\DynamicsCRMProvider\DynamicsCRMProvid

open System
open System.Linq
open FSharp.Data.TypeProviders

// configure the Dynamics CRM type provider
let xrm = XrmDataProvider<"http://server/org/XRMServices/
let accounts = xrm.accountSet |> Seq.toList

// now you have typed access to Dynamics CRM

let accounts =
    query { for a in xrm.accountSet do
            where (a.name.Contains "Contoso")
            select a } |> Seq.toList
```

Useful links ❤

- F# for fun and profit
- F# Software Foundation
- F# Guide
- Tour of F#
- Pluralsight

Thank you for your attention! 👍

Q&A + Discussion

Martin Hornáček

martin.hornacek@accenture.com