Writing F# Type Providers

Contents

[1 How to Write a Provider – Minimal Example 1](#_Toc266561465)

[1.1 Design 1](#_Toc266561466)

[1.2 Implementation 1](#_Toc266561467)

[1.3 Usage 1](#_Toc266561468)

[2 How to Write a Provider – A WMI Provider 1](#_Toc266561469)

[2.1 Design 1](#_Toc266561470)

[2.2 Example 2](#_Toc266561471)

[3 Going Further 2](#_Toc266561472)

[3.1 Adding Rich Metadata Experience 3](#_Toc266561473)

[3.2 Accessing Project-Local or Script-Local Resources 3](#_Toc266561474)

[3.3 Reporting Errors from Providers 3](#_Toc266561475)

[3.4 Invalidation 3](#_Toc266561476)

[3.5 Caching schema information 3](#_Toc266561477)

[3.6 Passing Static Parameters using Mangled Names 3](#_Toc266561478)

[3.7 Generative Providers 3](#_Toc266561479)

[4 Known Problems & Design Issues 3](#_Toc266561480)

[5 API Documentation 4](#_Toc266561481)

[6 Which DLL Name and Namespace Names should I use? 5](#_Toc266561482)

[7 Notes 8](#_Toc266561483)

[7.1 Data Providers 8](#_Toc266561484)

[7.1.1 CSV Files 8](#_Toc266561485)

[7.1.2 Dbml Files 8](#_Toc266561486)

[7.1.3 Contingency Table Files 9](#_Toc266561487)

[7.2 Web Provider 9](#_Toc266561488)

[7.3 Excel Provider 9](#_Toc266561489)

# Introduction

This doc covers the basics of how to write F# type providers, and some of the design issues associated with providers.

# How to Write a Provider – Minimal Example

This sample is “**HelloWorld.TypeProvider**” in the “Extenders” directory of the SDK.

## Design

The provider makes available a “type space” as follows (using F# signature syntax):

namespace HelloWorld

type HelloWorldType

static member StaticProperty1 : string

static member StaticProperty2 : int

Note the set of types provided is finite – this example doesn’t leverage any of the ability of providers to provide many (or infinite) types, not types dependent on a schema.

## Implementation

See HelloWorldProvider

## Usage

After building, the following script in the project directory runs correctly:

#r @"bin\Debug\HelloWorld.TypeProvider.dll"

if HelloWorld.HelloWorldType.StaticProperty1 = "You got a static property" then

printfn "Test passed"

if HelloWorld.HelloWorldType.StaticProperty2 = 42 then

printfn "Test passed"

# How to Write a Provider – A WMI Provider

## Design

The provider makes available a “type space” as follows (using F# signature syntax with meta notation):

namespace Microsoft.Management.Data

type LocalMachine =

*for each management object under \\localhost\root\cimv2 generate the following:*

static member *ObjectName* : seq<*ObjectName*\_Class>

type Classes =

*for each management object under \\localhost\root\cimv2 generate the following:*

type *ObjectName*\_Class =

inherit System.Management.ManagementObject

*for each subclass in the management object generate:*

member *SubclassName*+ : *TranslatedManagementType*

Note the set of types provided is dependent on the management object on the local machine. These objects are those returned by the following .NET code:

let getWmiClasses() =

let scope = ManagementScope(@"\\localhost\root\cimv2")

let path = ManagementPath("")

let mclass = new ManagementClass(scope, path, ObjectGetOptions())

let options = EnumerationOptions()

options.EnumerateDeep <- true

mclass.GetSubclasses(options)

Note that “companion” types are injected under LocalMachine.Classes. The user doesn’t normally reference these types textually.

## Example

Here is an example of using the provider:

#r @"..\Debug\bin\Microsoft.Management.Data.dll"

#r "System.Management.dll"

for proc in Microsoft.Management.Data.LocalMachine.Win32\_Process do

printfn "process name is '%s'" proc.Name

for bios in Microsoft.Management.Data.LocalMachine.Win32\_BIOS do

printfn "bios versions: '%A'" bios.BIOSVersion

# Going Further

## Adding Rich Metadata Experience

You can add XML Doc help to your returned System.Reflection objects by giving them an attribute. See uses of **ExtensionTypeXmlDocAttribute** in the API and code.

You can add definition locations to your returned System.Reflection objects by giving them an attribute. See uses of **DefinitionLocationAttribute** in the API and code.

## Accessing Project-Local or Script-Local Resources

Each instance of a type provider is given an ExtensionTypreResolverConfig. This contains the “resolution folder” for the provider, i.e. the project folder for the compilation or the directory containing a script.

## Reporting Errors from Providers

Providers can report errors by raising an **ExtensionTypeException** from **System.Runtime.CompilerServices**.

## Invalidation

Providers can raise invalidation signals to notify the F# Language Service that the schema assumptions may have changed. A re-typecheck will occur.

## Caching schema information

Providers will often need to cache access to schema information. The full architecture for this is still being decided. In the meantime, using adhoc techniques is reasonable, storing the information under “obj\...” in the resolution folder, or as user data.

## Passing Static Parameters using Mangled Names

Static parameters can be passed using name mangling, e.g.

type MainWindow = Designer.XamlFile.``MainWindow.xaml``

There are many discussions about whether a richer attribute-like syntax should be provided within static names. It is likely for V1 we will just stick to names.

## Generative Providers

TBD. See the “WebAPIs” example. These can be subtle

# Known Problems & Design Issues

The use of the [<Generate>] attribute is not yet required or checked.

See the separate doc. for known design issues and design notes. Please take the time to read this before submitting feedback.

# API Documentation

Here is the API for System.Runtime.CompilerServices (as of 10 July 2010)

using System;

using System.Reflection;

using System.Linq.Expressions;

namespace System.Runtime.CompilerServices

{

/// <summary>

/// Place on a class that implements IExtensionTypeResolver to extend the compiler.

/// </summary>

[AttributeUsage(AttributeTargets.Class, AllowMultiple = false)]

sealed public class CompilerExtensionAttribute : System.Attribute

{

public CompilerExtensionAttribute() { }

public bool IsGenerative { get; set; }

}

/// <summary>

/// Place attribute on runtime assembly to indicate that there is a corresponding compiler

/// extension assembly that consumes this runtime assembly. Runtime and designer assembly may be the same.

/// </summary>

[AttributeUsage(AttributeTargets.Assembly)]

sealed public class CompilerExtensionAssemblyAttribute : System.Attribute

{

private string assemblyName;

public CompilerExtensionAssemblyAttribute(string assemblyName)

{

this.assemblyName = assemblyName;

}

public string AssemblyName

{

get { return this.assemblyName; }

}

}

// Add this attribute to extension types, and methods and properties of extensions types to provide XML comments.

[AttributeUsage(AttributeTargets.Class | AttributeTargets.Method | AttributeTargets.Property, AllowMultiple = false)]

sealed public class ExtensionTypeXmlDocAttribute : System.Attribute

{

private string commentText;

public string CommentText

{

get { return this.commentText; }

}

public ExtensionTypeXmlDocAttribute(string comment)

{

this.commentText = comment;

}

}

[AttributeUsage(AttributeTargets.Class | AttributeTargets.Method | AttributeTargets.Property, AllowMultiple = false)]

sealed public class DefinitionLocationAttribute : System.Attribute

{

public DefinitionLocationAttribute() { }

public string FilePath { get; set; }

public int Line { get; set; }

public int Column { get; set; }

}

sealed public class ExtensionTypeResolverConfig

{

public ExtensionTypeResolverConfig(string resolutionFolder)

{

this.ResolutionFolder = resolutionFolder;

}

public string ResolutionFolder { get; private set; }

}

public interface IExtensionTypeResolver

{

/// <summary>

/// Namespace name the this ExtensionTypeResolver injects types into.

/// </summary>

string NamespaceName { get; }

/// <summary>

/// Return the injected types.

/// </summary>

/// <returns></returns>

Type[] GetTypes();

/// <summary>

/// Compilers call this method to query ExtensionTypeResolver for a type <c>name</c>.

/// </summary>

/// Resolver should return a type called <c>name</c> in namespace <c>NamespaceName</c> or <c>null</c> if the type is unknown.

/// <param name="name">short name of as type</param>

/// <param name="bindingAttr">binding flags, most importantly BindingFlags.IgnoreCase</param>

/// <returns></returns>

Type GetType(string name, BindingFlags bindingAttr);

/// <summary>

/// Called by the compiler to ask for an Expression tree to replace the given MethodBase with.

/// </summary>

/// <param name="syntheticMethodBase">MethodBase that was given to the compiler by a type returned by a GetType(s) call.</param>

/// <param name="parameters">Expressions that represent the parameters to this call.</param>

/// <returns>An Expression that the compiler will use in place of the given method base.</returns>

Expression GetInvokerExpression(MethodBase syntheticMethodBase, ParameterExpression[] parameters);

/// <summary>

/// Triggered when an assumption changes that invalidates the resolutions so far reported by the provider

/// </summary>

event System.EventHandler Invalidate;

}

public class ExtensionTypeException : Exception

{

public ExtensionTypeException(string message) : base(message) { }

}

}

# Which DLL Name and Namespace Names should I use?

The table below can act as a guide for our recommended naming for provider DLLs and namespaces.

Naming is very, very important for this work. Please take the time to look at the examples below. Not all of our samples yet follow this naming convention, and the naming convention may certainly undergo further change. We expect provider writer to keep up with these changes ☺

In general most provider DLLs should end in “TypeProvider”. However there are some exceptions when working with a single data source where the user is very unlikely to actually “use” the types in provided, they are just accessing data (e.g. Freebase or WMI)

// Naming conventions for BCL-shipped DLLs providing connections to

// formats. We don’t expect these to ship in dev11, hence in grey.

// System.[Data|Designer|Web].<technology>.Provider.dll

#r "System.Data.Linq.SqlTypeProvider.dll"

#r "System.Data.TextTableTypeProvider.dll" (contains Csv, TextTable, ...)

#r "System.Data.XmlTypeProvider.dll"

#r "System.Data.OdbcTypeProvider.dll"

#r "System.Xaml.XamlTypeProvider.dll"

#r "System.Web.ServiceTypeProvider.dll"

// Naming conventions for Microsoft-shipped DLLs providing

// connections to proprietary formats. We don’t expect these to

// ship these ourselves in dev11, hence in grey.

// <company>.<technology>.<others>.dll

#r "System.Management.TypeProvider.dll" // WMI data

#r "Microsoft.SharePoint.TypeProvider.dll"

#r "Microsoft.Excel.TablesTypeProvider.dll"

#r "Microsoft.Win32.Registry.TypeProvider.dll"

// Naming conventions for 3rd party DLLs providing connections to own sources.

// We won’t ship these in dev11, hence in grey.

// <company>.<technology>.<others>.dll

#r "Freebase.TypeProvider.dll"

#r "Wikipedia.TypeProvider.dll"

#r "Yahoo.Data.Directory.TypeProvider dll"

#r "Google.Data.Directory.TypeProvider.dll"

#r "Microsoft.Data.Directory.TypeProvider.dll"

#r "ProgrammableWeb.Services.Directory.TypeProvider.dll"

#r "Wsindex.Services.Directory.TypeProvider.dll"

// DLL conventions for incubation DLLs providing connections

// to open file formats and 3rd party sources. We would expect to ship

// some of these on CodePlex etc.

#r "Microsoft.Data.Linq.SqlTypeProvider.dll"

#r "Microsoft.Data.Entity.EntityTypeProvider.dll"

#r "Microsoft.Data.CsvTypeProvider.dll" (contains Csv, TextTable, ...)

#r "Microsoft.Data.TextTableTypeProvider.dll" (contains Csv, TextTable, ...)

#r "Microsoft.Data.ResxTypeProvider.dll"

#r "Microsoft.Data.OdbcTypeProvider.dll"

#r "Microsoft.Xml.XmlTypeProvider.dll"

#r "Microsoft.Xaml.XamlTypeProvider.dll"

#r "Microsoft.Web.ServicesTypeProvider.dll"

#r "Microsoft.Management.TypeProvider.dll"

#r "Microsoft.SharePoint.TypeProvider.dll"

#r "Microsoft.Excel.ExcelTablesTypeProvider.dll"

// codeplex samples

#r "Web.DataStore.Freebase.dll"

#r "Web.DataStore.Wikipedia.dll"

#r "Web.Directory.Yahoo.dll"

#r "Web.Directory.Google.dll"

#r "Web.Directory.ProgrammableWeb.dll"

#r "Web.Directory.Wsindex.dll"

#r "Microsoft.FSharp.JavaImporter.dll"

#r "Microsoft.FSharp.JavascriptImporter.dll"

#r "Microsoft.FSharp.RImporter.dll"

#r "Microsoft.FSharp.MImporter.dll"

#r "Microsoft.FSharp.PythonImporter.dll"

// Namespace conventions for Fabrikom incubation DLLs providing connections

// to open file formats and 3rd party sources.

#r "Fabrikom.Data.Linq.SqlTypeProvider.dll"

...

// Namespace conventions for Microsoft or 3rd party components

// providing connections to own proprietary data. We don’t expect these to

// ship these ourselves in dev11, hence in grey.

Microsoft.SharePoint.Data.<???>

Yahoo.Data.<dataset name>

Google.Data.<dataset name>

// Namespace conventions for Fabrikom DLLs providing connections

// to open file formats and 3rd party sources. We would expect to ship

// some of these.

...

Fabrikom.Data.OdbcTypeProvider.OdbcConnection.<dataset name>

// Namespace conventions for our components

Microsoft.Data.OdbcTypeProvider.OdbcConnection.<connection string>

Microsoft.Data.CsvTypeProvider.CsvFile.<file uri>

Microsoft.Data.ContingencyTableTypeProvider.ContingencyTableFile.<file uri>

Microsoft.Data.ResxTypeProvider.ResxFile.<file uri>

Microsoft.Data.TextTableTypeProvider.TextTableFile.<file uri + parameters>

?columnNames="a, b"

?columnClasses="..."

?rowNames="a,b"

?encoding="..."

?header=true/false

?sep=' '

?quoting='\"'

?na.strings='NA na n/a'

?strip.white=true/false

?blank.lines.skip =true/false

?comment.char='#'

?allowEscapes=true/false

Microsoft.Xml.XmlTypeProvider.XmlFile.<file uri>

Microsoft.Xml.XmlTypeProvider.XsdFile.<file uri>

Microsoft.Xaml.XamlTypeProvider.XamlFile.<file uri>

Microsoft.Data.Linq.SqlTypeProvider.DbmlFile.<file uri>

Microsoft.Data.Linq.SqlTypeProvider.DirectConnection.<connection string>

Microsoft.Data.Linq.SqlTypeProvider.ServerDirectory.<server name>.<db name>

Microsoft.Data.Entity.EntityTypeProvider.DirectConnection.<connection string>

Microsoft.Data.Entity.EntityTypeProvider.EdmxFile.<connection string>

Microsoft.Data.Entity.EntityTypeProvider.ServerDirectory.<server name>.<db name>

Microsoft.Management.Data.AnyMachine.<wmi name>

Microsoft.Management.Data.LocalMachine.<wmi name>

Microsoft.Management.Data.NamedMachine.<machine name>.<wmi name>

#r "foo.odata"

#r "foo.dbml"

#r "foo.wsdl"

Microsoft.Web.ServicesTypeProvider.ServiceUri.<service uri>

Microsoft.Web.ServicesTypeProvider.WsdlFile.<file name>

// codeplex samples

Web.DataStore.Freebase.<domain>

Web.DataStore.Wikipedia.<domain>

Web.Directory.Yahoo.<category name>.<service name>.<view name>

Web.Directory.Google.<category name>.<service name>.<view name>

Web.Directory.ProgrammableWeb.<category name>.<service name>.<view name>

Web.Directory.Wsindex.<category name>.<service name>.<view name>

#r "javaStuff.jar"

#load "jquery.js"

#load "someFile.r"

#load "someFile.m"

#load "someFile.py"

// User code when “open” is taken into account

open Microsoft.Data.OdbcTypeProvider

open Microsoft.Data.CsvTypeProvider

open Microsoft.Data.ResxTypeProvider

OdbcConnection.<connection string>

CsvFile.<file uri>

[<Generate>] namespace MyResxFile = ResxFile.<file uri>

[<Generate>] namespace MyService1 = ServiceUri.<service uri>

[<Generate>] namespace MyService2 = WsdlFile.<file name>

[<Generate>] namespace Northwnd = DbmlFile.``northwnd.dbml``

let db = new Northwnd(<conn-string>)

# Notes

## Data Providers

### CSV Files

### Dbml Files

Namespace is in file

### Contingency Table Files

#### Example

R standard data set UCBAdmissions which is a 3-dimensional contingency table resulting from classifying applicants to graduate school at UC Berkeley for the six largest departments in 1973 classified by admission and sex.

Dept A B C D E F

Admit Gender

Admitted Male 512 353 120 138 53 22

Female 89 17 202 131 94 24

Rejected Male 313 207 205 279 138 351

Female 19 8 391 244 299 317

## Web Provider

## Excel Provider