免费电子书

学习

xamarin

Free unaffiliated eBook created from
Stack Overflow contributors.

#xamarin

1: xamarin

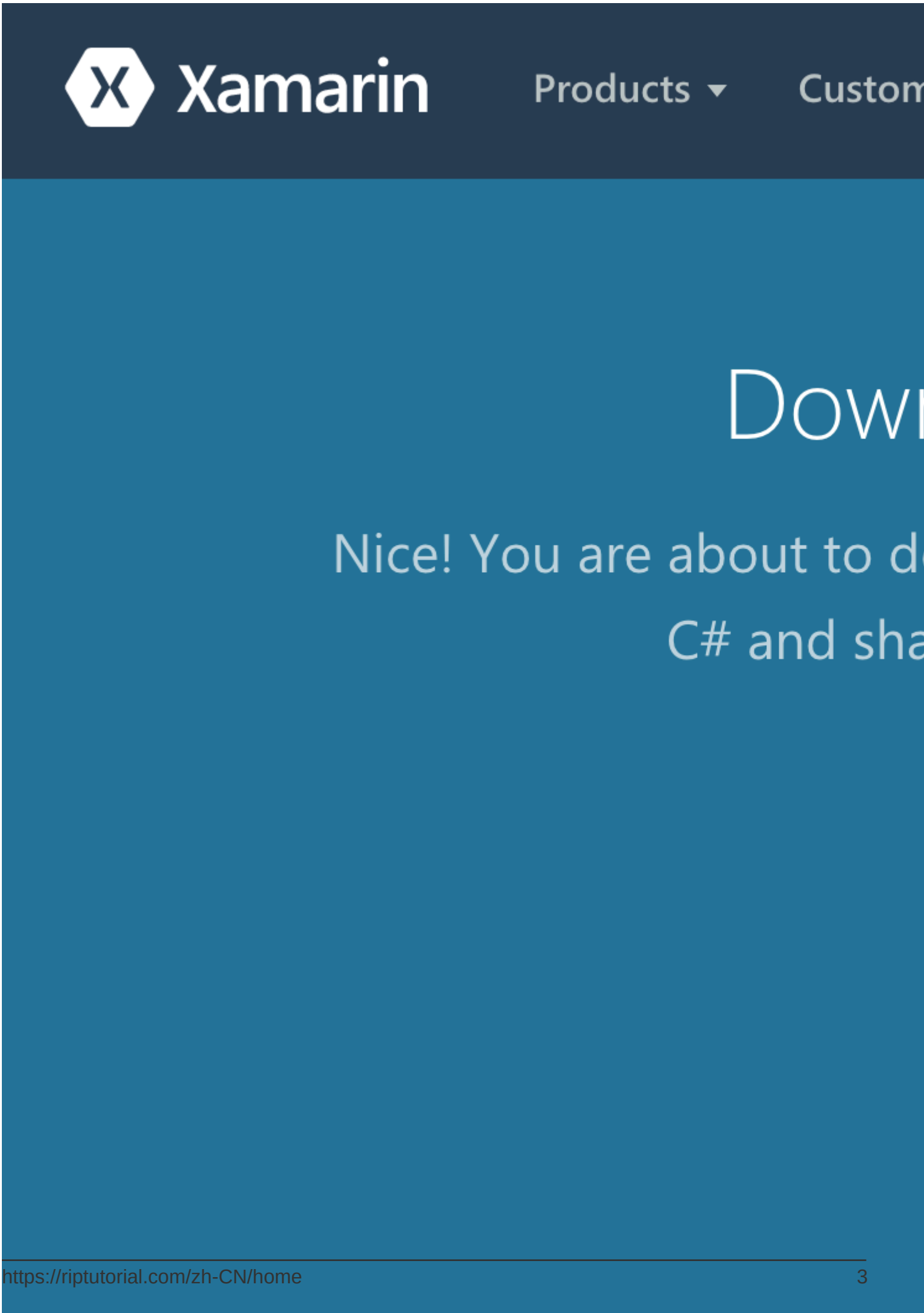
xamarin。

xamarin。 xamarin。

Examples

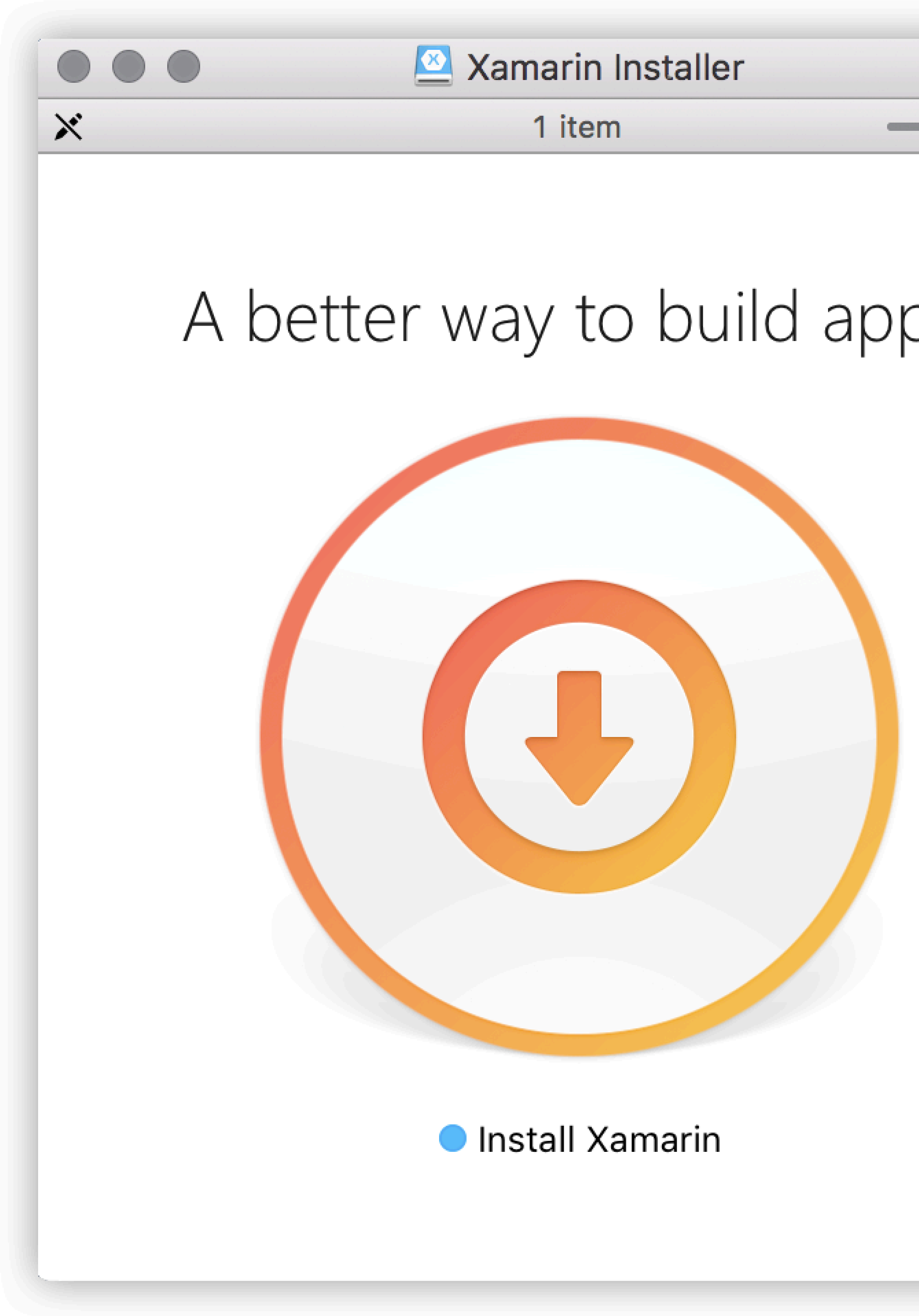
OS XXamarin Studio

OS XXamarinXamarin Studio Community。 。


Xamarin
Products
Custom
Dow
Nice! You are about to d
C# and sha

- Mac App StoreApple Developer WebsiteXcode。
- Mac OS X10.10

XamarinXamarin。



OS X Gatekeeper “”

2: Annotations

mvc.net。Xamarin

Examples

nuget`System.ComponentModel.Annotations`

```
public class BankAccount
{

   public enum AccountType
   {
       Saving,
       Current
   }

   [Required(ErrorMessage="First Name Required")]
   [MaxLength(15,ErrorMessage="First Name should not more than 1`5 character")]
   [MinLength(3,ErrorMessage="First Name should be more than 3 character")]
   public string AccountHolderFirstName { get; set; }

   [Required(ErrorMessage="Last Name Required")]
   [MaxLength(15,ErrorMessage="Last Name should not more than 1`5 character")]
   [MinLength(3,ErrorMessage="Last Name should be more than 3 character")]
   public string AccountHolderLastName { get; set; }

   [Required]
   [RegularExpression("^[0-9]+$", ErrorMessage = "Only Number allowed in AccountNumber")]
   public string AccountNumber { get; set; }

   public AccountType AcType { get; set; }
}
```

```
public class GenericValidator
{
    public static bool TryValidate(object obj, out ICollection<ValidationResult> results)
    {
        var context = new ValidationContext(obj, serviceProvider: null, items: null);
        results = new List<ValidationResult>();
        return Validator.TryValidateObject(
            obj, context, results,
            validateAllProperties: true
        );
    }
}
```

```
var bankAccount = new BankAccount();
ICollection<ValidationResult> lstvalidationResult;

bool valid = GenericValidator.TryValidate(bankAccount, out lstvalidationResult);
if (!valid)
{
    foreach (ValidationResult res in lstvalidationResult)
```

```
    {
        Console.WriteLine(res.MemberNames +":"+ res.ErrorMessage);
    }

}
Console.ReadLine();
```

```
First Name Required
Last Name Required
The AccountNumber field is required.
```

Annotations https://riptutorial.com/zh-CN/xamarin/topic/9720/annotations

3:

Examples

Bridge。Xamarin。Portable Class LibraryXamarin FormsAndroidAlertDialog。AlertDialog。

```
// Define a common interface for the behavior you want in your common project (Forms/Other
PCL)
public interface IPlatformReporter
{
    string GetPlatform();
}


// In Android/iOS/Win implement the interface on a class
public class DroidReporter : IPlatformReporter
{
    public string GetPlatform()
    {
        return "Android";
    }
}


public class IosReporter : IPlatformReporter
{
    public string GetPlatform()
    {
        return "iOS";
    }
}


// In your common project (Forms/Other PCL), create a common class to wrap the native
implementations
public class PlatformReporter : IPlatformReporter
{
    // A function to get your native implemenation
    public static func<IPlatformReporter> GetReporter;

    // Your native implementation
    private IPlatformReporter _reporter;

    // Constructor accepts native class and stores it
    public PlatformReporter(IPlatformReporter reporter)
    {
        _reporter = GetReporter();
    }

    // Implement interface behavior by deferring to native class
    public string GetPlatform()
    {
        return _reporter.GetPlatform();
    }
}
```

```
// In your native code (probably MainActivity/AppDelegate), you just supply a function that
returns your native implementation
public class MainActivity : Activity
{
    protected override void OnCreate(Bundle bundle)
    {
        base.OnCreate(bundle);
        SetContentView(Resource.Layout.activity_main);

        PlatformReporter.GetReporter = () => { return new DroidReporter(); };
    }
}


public partial class AppDelegate : UIApplicationDelegate
{
    UIWindow window;

    public override bool FinishedLaunching(UIApplication app, NSDictionary options)
    {
        window = new UIWindow(UIScreen.MainScreen.Bounds);
        window.RootViewController = new UIViewController();
        window.MakeKeyAndVisible();

        PlatformReporter.GetReporter = () => { return new IosReporter(); };

        return true;
    }
}


// When you want to use your native implementation in your common code, just do as follows:
public void SomeFuncWhoCares()
{
    // Some code here...

    var reporter = new PlatformReporter();
    string platform = reporter.GetPlatform();

    // Some more code here...
}
```

Service Locator。Bridge Pattern。 -。

```
// Define a service locator class in your common project
public class ServiceLocator {
    // A dictionary to map common interfaces to native implementations
    private Dictionary<object, object> _services;

    // A static instance of our locator (this guy is a singleton)
    private static ServiceLocator _instance;

    // A private constructor to enforce the singleton
    private ServiceLocator() {
        _services = new Dictionary<object, object>();
    }

    // A Singleton access method
    public static ServiceLocator GetInstance() {
        if(_instance == null) {
```

```
            _instance = new ServiceLocator();
        }

        return _instance;
    }

    // A method for native projects to register their native implementations against the
common interfaces
    public static void Register(object type, object implementation) {
        _services?.Add(type, implementation);
    }

    // A method to get the implementation for a given interface
    public static T Resolve<T>() {
        try {
            return (T) _services[typeof(T)];
        } catch {
            throw new ApplicationException($"Failed to resolve type: {typeof(T).FullName}");
        }
    }


//For each native implementation, you must create an interface, and the native classes
implementing that interface
public interface IA {
    int DoAThing();
}


public interface IB {
    bool IsMagnificent();
}


public class IosA : IA {
    public int DoAThing() {
        return 5;
    }
}


public class DroidA : IA {
    public int DoAThing() {
        return 42;
    }
}


// You get the idea...


// Then in your native initialization, you have to register your classes to their interfaces
like so:
public class MainActivity : Activity
{
    protected override void OnCreate(Bundle bundle)
    {
        base.OnCreate(bundle);
        SetContentView(Resource.Layout.activity_main);

        var locator = ServiceLocator.GetInstance();
```

```
        locator.Register(typeof(IA), new DroidA());
        locator.Register(typeof(IB), new DroidB());
    }
}


public partial class AppDelegate : UIApplicationDelegate
{
    UIWindow window;

    public override bool FinishedLaunching(UIApplication app, NSDictionary options)
    {
        window = new UIWindow(UIScreen.MainScreen.Bounds);
        window.RootViewController = new UIViewController();
        window.MakeKeyAndVisible();

        var locator = ServiceLocator.GetInstance();
        locator.Register(typeof(IA), new IosA());
        locator.Register(typeof(IB), new IosB());

        return true;
    }
}


// Finally, to use your native implementations from non-native code, do as follows:
public void SomeMethodUsingNativeCodeFromNonNativeContext() {
     // Some boring code here


     // Grabbing our native implementations for the current platform
     var locator = ServiceLocator.GetInstance();
     IA myIA = locator.Resolve<IA>();
     IB myIB = locator.Resolve<IB>();


    // Method goes on to use our fancy native classes
}
```

https://riptutorial.com/zh-CN/xamarin/topic/6183/

S. No		Contributors
1	xamarin	Akshay Kulkarni, Community, Gil Sand, hankide, Joel Martinez, Marius Ungureanu, Sven-Michael Stübe, thomasvdb
2	Annotations	Niek de Gooijer
3		kellen lask, valdetero