

 무료 전자 책

배우기

xamarin

Free unaffiliated eBook created from  
**Stack Overflow contributors.**

#xamarin

.....	1
<b>1: xamarin</b> .....	<b>2</b>
.....	2
Examples.....	2
OS X Xamarin Studio .....	2
.....	4
.....	5
Xamarin Studio Hello World : Xamarin.Forms.....	5
<b>2:</b> .....	<b>6</b>
.....	6
Examples.....	6
.....	6
<b>3:</b> .....	<b>8</b>
Examples.....	8
.....	8
.....	9
.....	12

---

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: [xamarin](#)

It is an unofficial and free xamarin ebook created for educational purposes. All the content is extracted from [Stack Overflow Documentation](#), which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official xamarin.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to [info@zzzprojects.com](mailto:info@zzzprojects.com)

---

# 1: xamarin

xamarin , xamarin .

xamarin . xamarin .

## Examples

### OS X Xamarin Studio

OS X Xamarin      Xamarin Studio Community . .



# Down

Nice! You are about to d

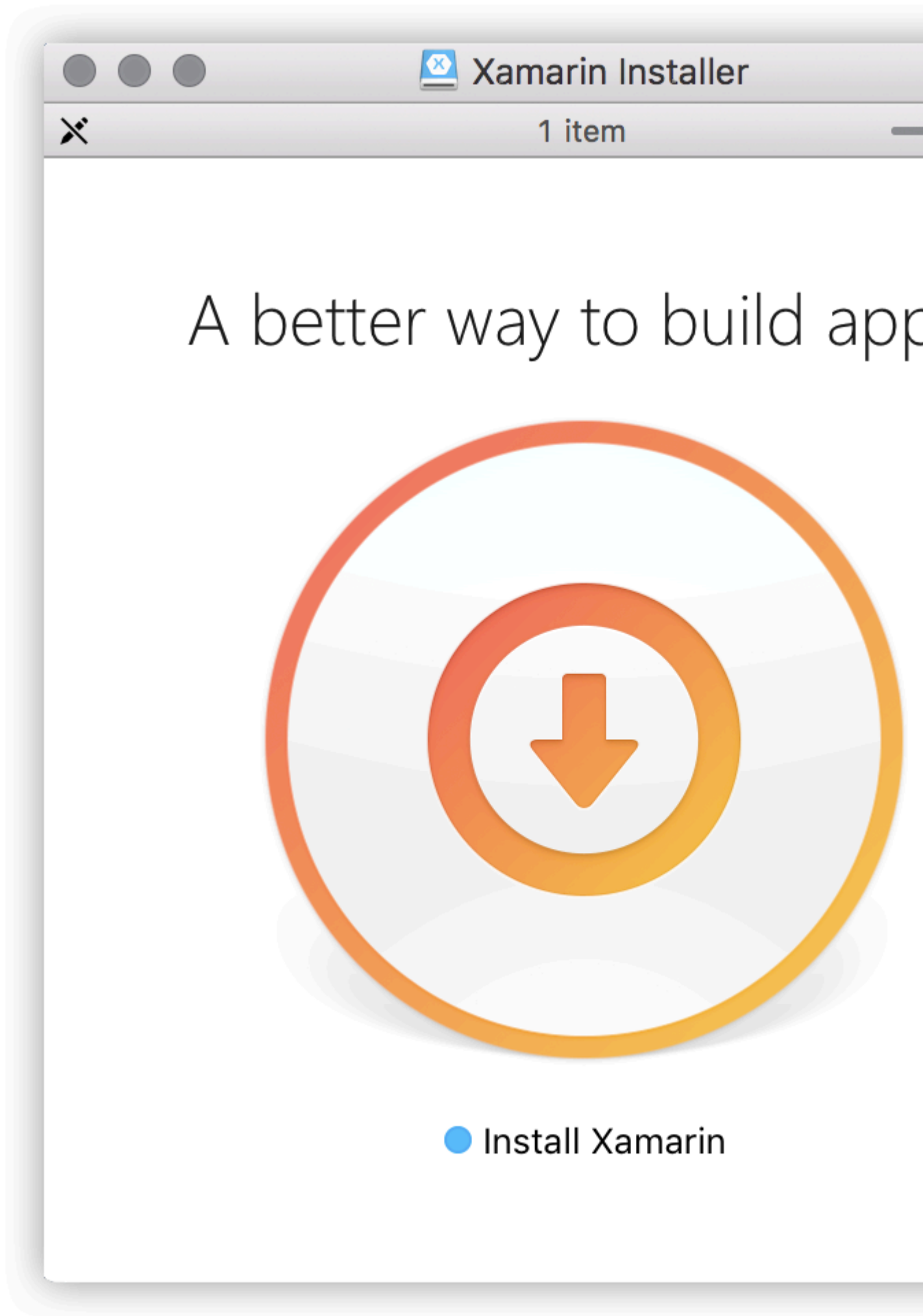
C# and sha

: Android SDK) .

Xamarin.iOS .

- [iOS](#) iOS SDK
- Mac App Store [Apple Developer Website](#) Xcode .
- Mac OS X Yosemite (10.10)

Xamarin Xamarin Installer .



## 2:

mvc.net . Xamarin .

## Examples

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
        );
    }
}
```

validator :

```
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.
```

: <https://riptutorial.com/ko/xamarin/topic/9720/---->

# 3:

## Examples

Bridge Inversion of Control . Xamarin : Xamarin Forms Android AlertDialog .  
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

```

// 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/ko/xamarin/topic/6183/---](https://riptutorial.com/ko/xamarin/topic/6183/---)

---

S. No		Contributors
1	xamarin	<a href="#">Akshay Kulkarni</a> , <a href="#">Community</a> , <a href="#">Gil Sand</a> , <a href="#">hankide</a> , <a href="#">Joel Martinez</a> , <a href="#">Marius Ungureanu</a> , <a href="#">Sven-Michael Stübe</a> , <a href="#">thomasvdb</a>
2		<a href="#">Niek de Gooijer</a>
3		<a href="#">kellen lask</a> , <a href="#">valdetero</a>