EBook Gratuito

APPRENDIMENTO xamarin

Free unaffiliated eBook created from
Stack Overflow contributors.

#xamarin

Sommario

Di

Capitolo 1: Iniziare con xamarin

Osservazioni

Questa sezione fornisce una panoramica su cosa sia xamarin e perché uno sviluppatore possa volerlo utilizzare.

Dovrebbe anche menzionare tutti i soggetti di grandi dimensioni all'interno di xamarin e collegarsi agli argomenti correlati. Poiché la documentazione di xamarin è nuova, potrebbe essere necessario creare versioni iniziali di tali argomenti correlati.

Examples

Installazione di Xamarin Studio su OS X

Il primo passo per avviare lo sviluppo di Xamarin su una macchina OS X, è scaricare e installare la versione della community di Xamarin Studio dal sito web ufficiale . Alcuni campi devono essere riempiti per scaricare il programma di installazione come mostrato nella figura qui sotto.

Dowr

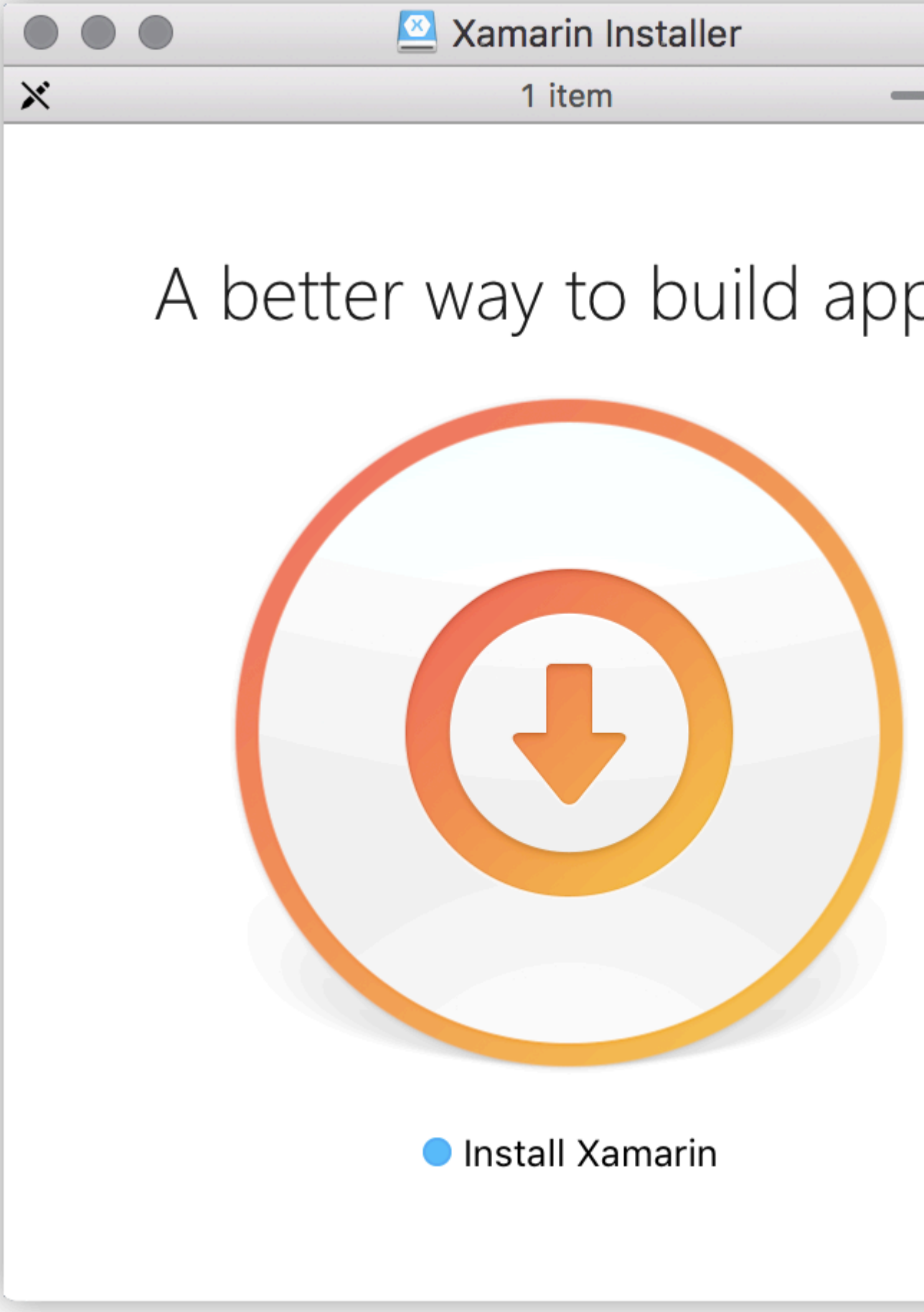
Nice! You are about to d

C# and sha

- L'ultima versione di Xcode dal Mac App Store o dal sito Web degli sviluppatori Apple .
  sito Web degli sviluppatori Apple .
- Mac OS X Yosemite (10.10) e versioni successive

Processo di installazione

Una volta soddisfatti i prerequisiti, esegui Xamarin Installer facendo doppio clic sul logo Xamarin.



~~OS X Gatekeeper può mostrare una finestra di dialogo che chiede conferma per aprire l'applicazi~~

Capitolo 2: Condivisione del codice tra i progetti

Examples

Il modello del ponte

Il pattern Bridge è uno degli schemi di progettazione di Inversion of Control più basilari. Per Xamarin, questo modello viene utilizzato per fare riferimento al codice dipendente dalla piattaforma da un contesto indipendente dalla piattaforma. Ad esempio: utilizzando AlertDialog di Android da una libreria di classi portatile o moduli Xamarin. Nessuno di quei contesti sa cosa sia un oggetto AlertDialog, quindi è necessario inserirlo in una casella affinché possano essere utilizzati.

```
// 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...
}
```

Il modello di localizzazione del servizio

Il modello di progettazione di Service Locator è un'iniezione molto vicina alla dipendenza. Come il Bridge Pattern, questo modello può essere utilizzato per fare riferimento al codice dipendente dalla piattaforma da un contesto indipendente dalla piattaforma. Più interessante, questo modello

si basa sul modello singleton: tutto ciò che inserisci nel localizzatore di servizio sarà un defacto singleton.

```
// 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
}
```

Leggi Condivisione del codice tra i progetti online:

https://riptutorial.com/it/xamarin/topic/6183/condivisione-del-codice-tra-i-progetti

Capitolo 3: Convalida dell'oggetto per annotazioni

introduzione

mvc.net introduce anotations di dati per la validazione del modello. Questo può essere fatto anche in Xamarin

Examples

Semplice esempio

Aggiungi il pacchetto nuget `System.ComponentModel.Annotations`

Definisci una classe:

```
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; }
}
```

Definire un validatore:

```
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
        );
    }
}
```

usa il validatore:

```
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();
```

Output generato:

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

Leggi Convalida dell'oggetto per annotazioni online:
https://riptutorial.com/it/xamarin/topic/9720/convalida-dell-oggetto-per-annotazioni

Titoli di coda

S. No	Capitoli	Contributors
1	Iniziare con xamarin	Akshay Kulkarni, Community, Gil Sand, hankide, Joel Martinez, Marius Ungureanu, Sven-Michael Stübe, thomasvdb
2	Condivisione del codice tra i progetti	kellen lask, valdetero
3	Convalida dell'oggetto per annotazioni	Niek de Gooijer