



Laboratorio

Bibliotecas de Clases Portables *(Portable Class Libraries)*

Versión: 1.0.0
Abril de 2017



[Miguel Muñoz Serafín](#)
@msmdotnet





CONTENIDO

INTRODUCCIÓN

EJERCICIO 1: CREANDO PROYECTOS DE BIBLIOTECAS MULTIPLATAFORMA REUTILIZABLES

Tarea 1. Crear un Proyecto PCL.

Tarea 2. Trabajando con el proyecto PCL

Tarea 3. Utilizando el proyecto PCL.

EJERCICIO 2: VALIDANDO TU ACTIVIDAD

Tarea 1. Agregar los componentes de la Capa de acceso a Servicio.

Tarea 2. Agregar la funcionalidad para validar la actividad.

Tarea 3. Ejecutar la aplicación.

RESUMEN



Introducción

Las Bibliotecas de Clases Portables (**Portable Class Libraries o PCL**) nos permiten escribir código y crear bibliotecas que pueden ser compartidas entre múltiples plataformas, incluyendo Xamarin.iOS, Xamarin.Android y Windows. Las Bibliotecas de Clases Portables se pueden crear en Xamarin Studio o Visual Studio y ser referenciadas en cada proyecto específico de la plataforma para permitir que el código pueda ser fácilmente compartido.

El soporte a las Bibliotecas de Clases Portables fue agregado en Xamarin.Android 4.10.1, Xamarin.iOS 7.0.4 y Xamarin Studio 4.2. Los proyectos PCL se habilitan automáticamente en Xamarin Studio en OS X y están incorporados en Visual Studio 2013 y posteriores.

Este laboratorio te permitirá conocer la forma en que funcionan las Bibliotecas de Clases Portables, así como la forma de crearlas y consumirlas en aplicaciones móviles multiplataforma.

Objetivos

Al finalizar este laboratorio, los participantes serán capaces de:

- Crear un Proyecto de Bibliotecas de Clases Portables PCL.
- Utilizar un Proyecto PCL desde un proyecto de aplicación.

Requisitos

Para la realización de este laboratorio es necesario contar con lo siguiente:

- Un equipo de desarrollo con Visual Studio. Los pasos descritos en este laboratorio fueron diseñados con Visual Studio Enterprise 2017 y Windows 10 Professional.
- Xamarin para Visual Studio.

Tiempo estimado para completar este laboratorio: **60 minutos**.



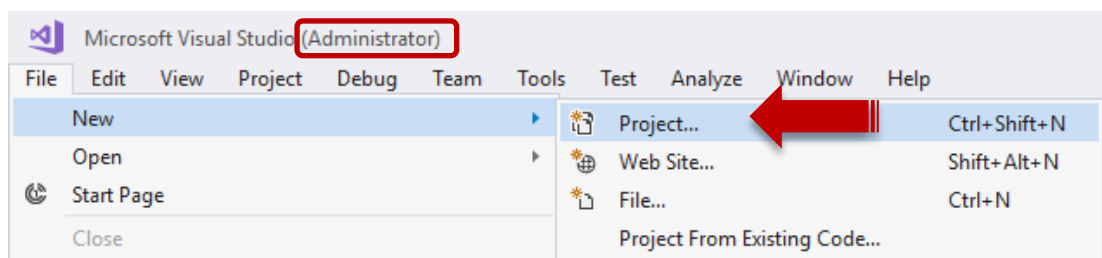
Ejercicio 1: Creando proyectos de bibliotecas multiplataforma reutilizables

En este ejercicio crearás una biblioteca de clases portable con Visual Studio que será referenciado por otros proyectos de aplicación.

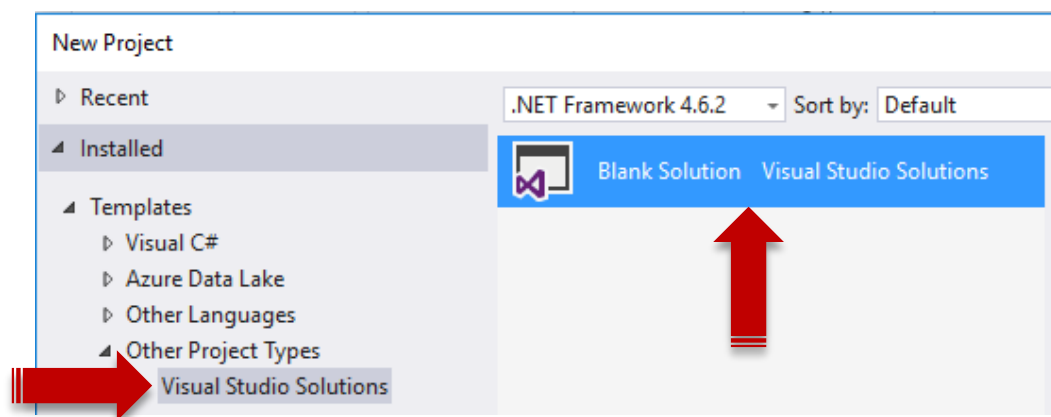
Tarea 1. Crear un Proyecto PCL.

Realiza los siguientes pasos para crear una solución con un proyecto PCL.

1. Abre Visual Studio en el contexto del Administrador.
2. Selecciona la opción **File > New > Project**.



3. En la ventana **New Project** selecciona la plantilla **Blank Solution**.



4. Proporciona el nombre y ubicación de la solución.

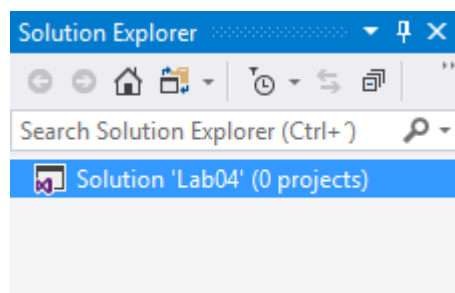


Name: Lab04
Location: C:\demos\
Solution name: Lab04

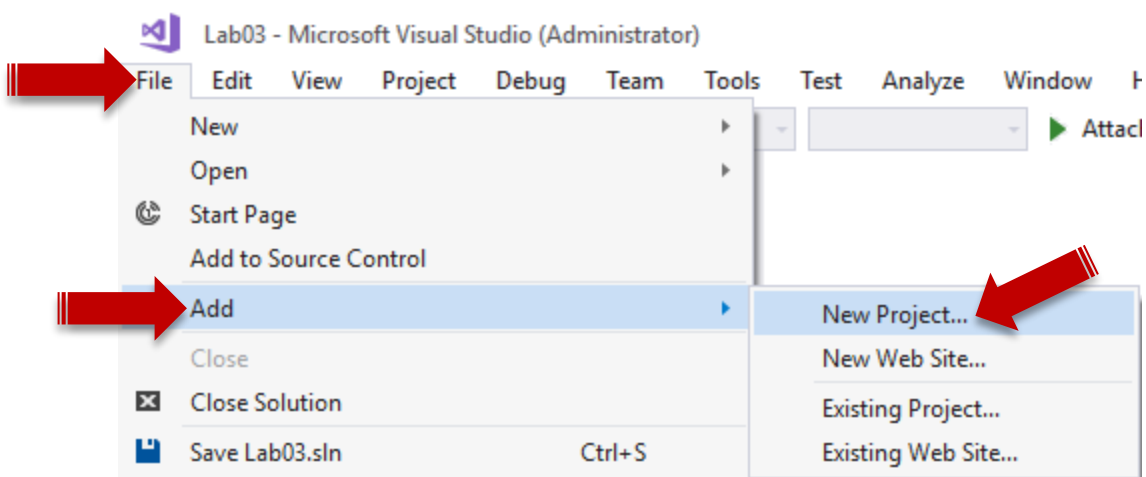
5. Haz clic en **OK** para crear la solución en blanco.

Browse...
☒ Create directory for solution
☐ Add to Source Control
OK Cancel

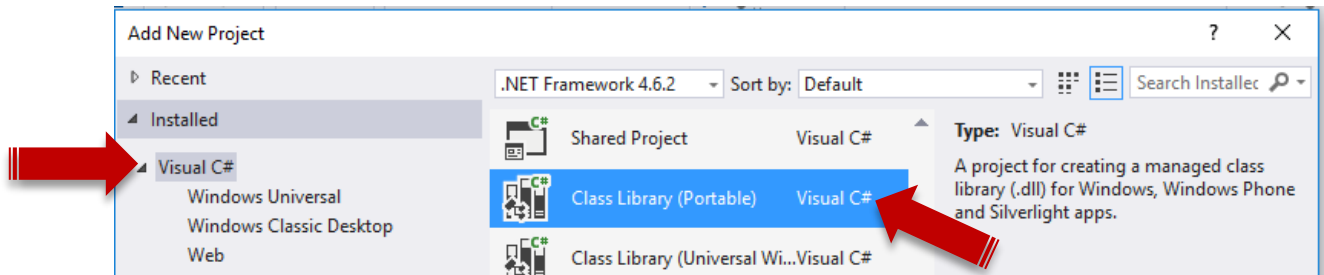
El explorador de soluciones de Visual Studio mostrará la solución vacía.



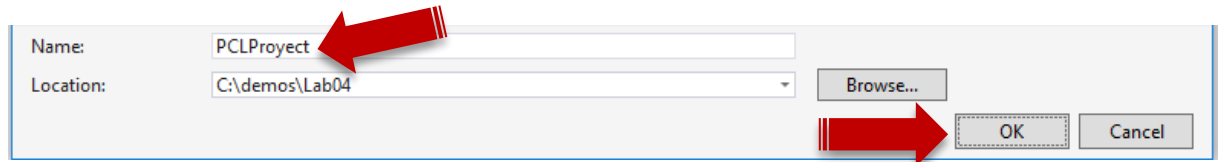
6. Para agregar un nuevo Proyecto PCL a la solución, selecciona la opción **File > Add > New Project...**



7. En la ventana **New Project** selecciona la plantilla **Class Library (Portable)**.



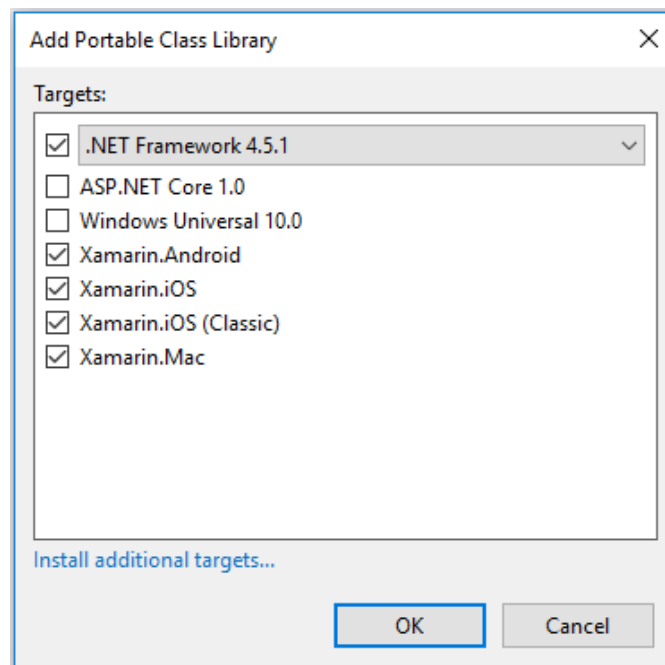
8. Asigna un nombre al proyecto PCL y haz clic en **OK** para agregarlo a la solución.



Agregar un proyecto PCL a una solución en Visual Studio es ligeramente diferente a agregar un proyecto de otro tipo.

Al crear una Biblioteca de Clases Portable, podemos elegir una combinación de plataformas en las que deseamos que se ejecute el código. Las opciones de compatibilidad que seleccionamos al crear una Biblioteca de Clases Portable son trasladadas en un identificador de "Perfil" que describe las plataformas que soporta la biblioteca.

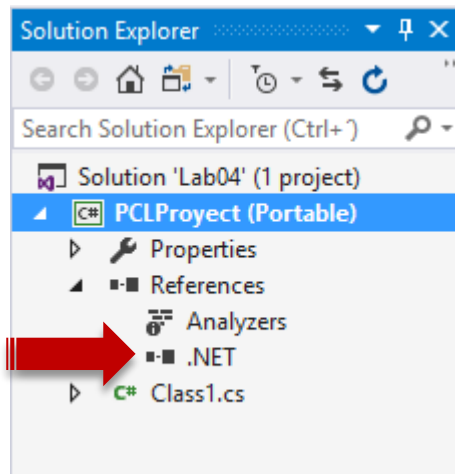
Visual Studio mostrará inmediatamente el cuadro de diálogo **Add Portable Class Library** para que el perfil se pueda configurar. En este cuadro de diálogo podemos seleccionar las plataformas que necesitamos dar soporte.





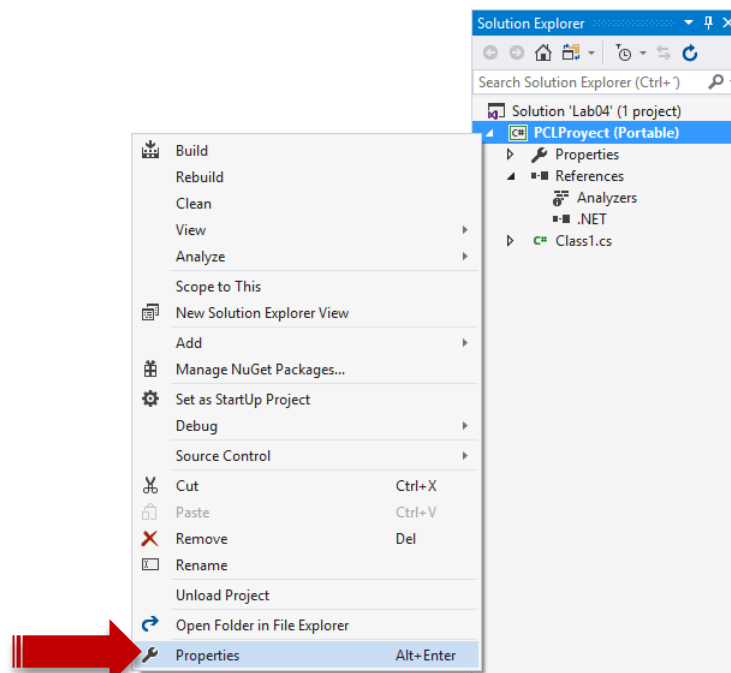
9. Haz clic en **OK** para aceptar las plataformas sugeridas y continuar.

El explorador de soluciones te mostrará el nuevo proyecto agregado. El nodo **References** indica que la biblioteca utiliza un subconjunto del .NET Framework (definido por el perfil PCL).



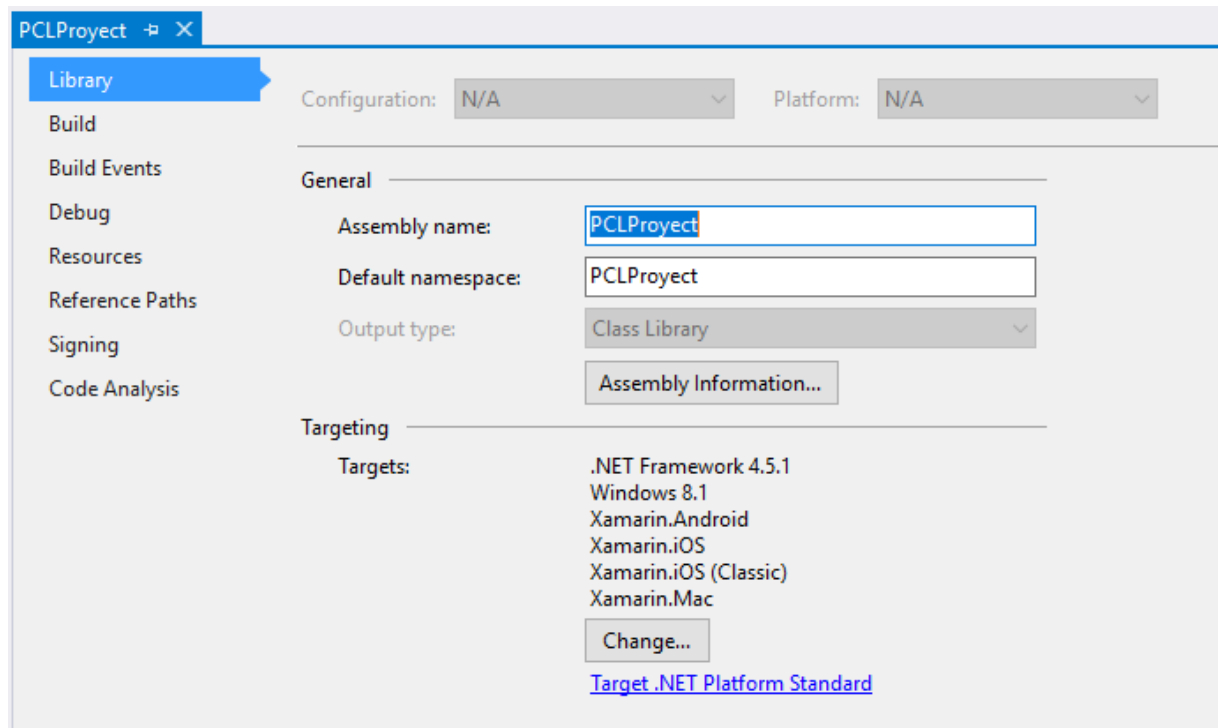
El proyecto PCL está listo para que podamos agregar el código a compartir. También puede ser referenciado por otros proyectos (proyectos de aplicación, proyectos de biblioteca e incluso otros proyectos PCL).

10. Selecciona la opción **Properties** del menú contextual del proyecto PCL para abrir su hoja de propiedades.





En la sección **Library** puedes observar y modificar la configuración del proyecto PCL. Puedes incluso actualizarlo a un tipo de proyecto **.NET Standard**.



Si se cambia el perfil después de que se haya agregado código al proyecto PCL, es posible que la biblioteca ya no compile si el código hace referencia a características que no forman parte del nuevo perfil seleccionado.

11. Cierra la hoja de propiedades del proyecto.

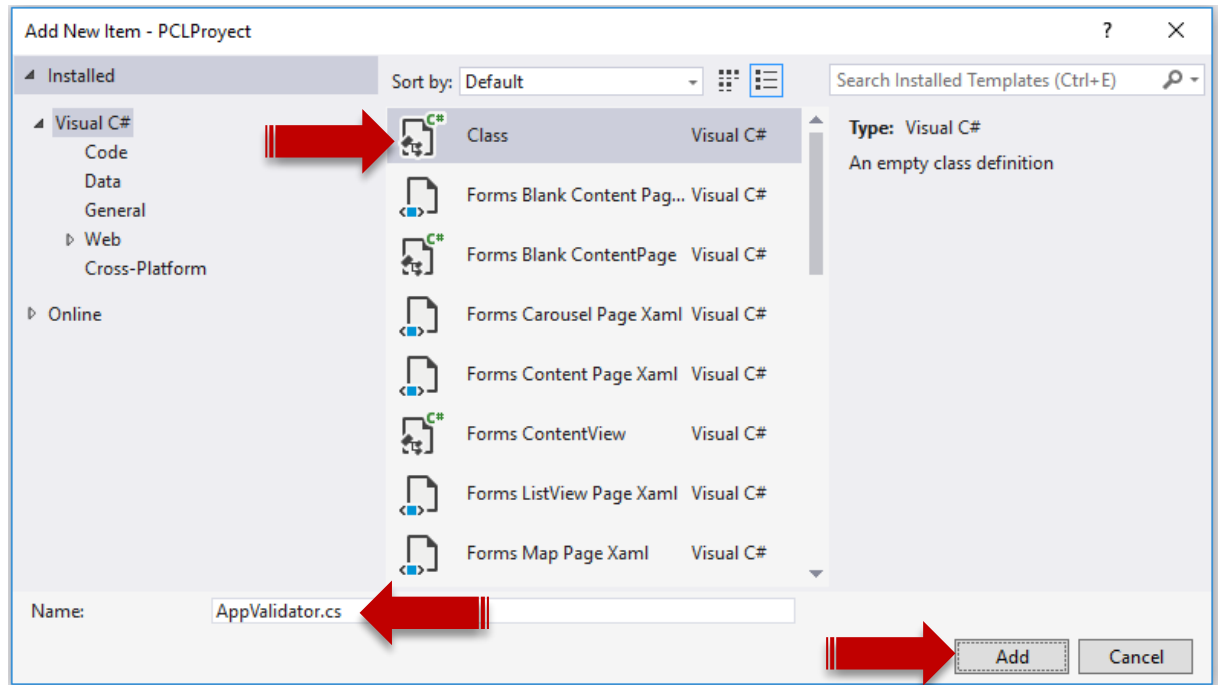
Tarea 2. Trabajando con el proyecto PCL.

Uno de los enfoques para soportar múltiples plataformas desde el mismo código base es la **Abstracción de plataforma**. La abstracción de plataforma se consigue utilizando interfaces o clases abstractas definidas en el código compartido e implementadas o extendidas en proyectos específicos de la plataforma. Escribir y extender el código compartido con abstracciones de clases es particularmente adecuado para bibliotecas de clases portables (PCL) porque tienen un subconjunto limitado del Framework disponible para ellos y no pueden contener directivas de compilación para soportar ramificaciones de código específicas de plataforma.

El uso de Interfaces o clases abstractas permite que las aplicaciones específicas de plataforma puedan implementar la interface o extender la clase para después ser pasadas a las bibliotecas compartidas como parámetros o propiedades y de esa manera aprovechar el código común.

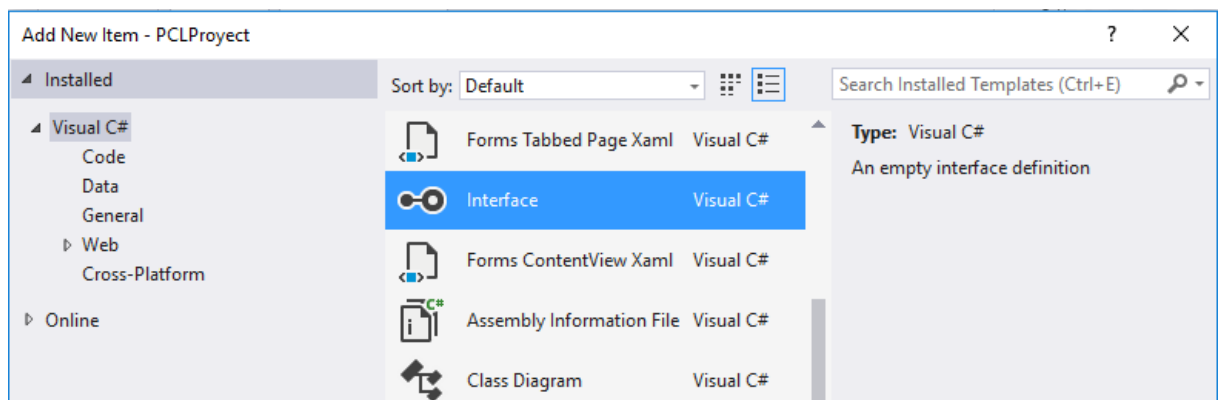


1. Elimina el archivo **Class1.cs** del proyecto PCL.
2. Agrega al proyecto PCL un nuevo archivo de clase llamado **AppValidator.cs**.



En este ejemplo, la clase **AppValidator** tendrá el código que compartiremos con otras plataformas. La clase implementará lógica de negocios para obtener un resultado que será mostrado en la pantalla del dispositivo actual. Debido a que el código compartido es ignorante de la plataforma en la que se estará ejecutando, necesitaremos un mecanismo para permitir que el código compartido ejecute la funcionalidad específica de la plataforma y esto lo realizaremos a través de la implementación de una interface que podrá ser inyectada al código compartido.

3. Agrega al proyecto PCL un nuevo elemento llamado **IDialog** utilizando la plantilla **Interface**.





4. Agrega el modificador **public** a la definición de la interface para que pueda ser accesible desde otros ensamblados.

```
public interface IDialog
{
}
```

5. Modifica el código de la interface **IDialog** para definir un miembro **Show** que permitirá a las aplicaciones específicas de plataforma implementar la funcionalidad para mostrar un mensaje en la pantalla, por ejemplo, con **AlertDialog** en Android o con **MessageDialog** en aplicaciones UWP.

```
public interface IDialog
{
    void Show(string message);
}
```

6. Abre el archivo **AppValidator.cs**.
7. Agrega el modificador **public** a la definición de la clase **AppValidator** para que pueda ser accesible desde otros ensamblados.

```
public class AppValidator
{
}
```

8. Agrega el siguiente código a la clase **AppValidator** para permitir al código cliente inyectar un objeto **IDialog** que ejecute el código específico de plataforma.

```
IDialog Dialog;
public AppValidator(IDialog platformDialog)
{
    Dialog = platformDialog;
}
```

9. Agrega el siguiente código al cuerpo de la clase.

```
public string Email { get; set; }
public string Password { get; set; }
public string Device { get; set; }

public void Validate()
{
    string Result;
    /* Aquí se puede implementar la funcionalidad
    principal de la clase. Por el momento solo devuelve
    una cadena fija. */

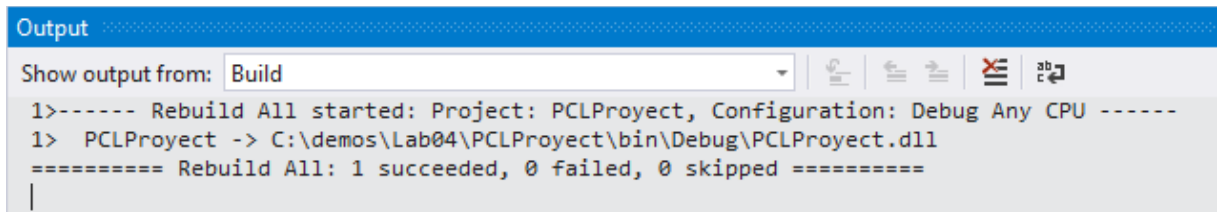
    Result = "¡Aplicación validada!";

    /* Invocar el código específico de la plataforma */
    Dialog.Show(Result);
}
```



El método **Validate** implementa lógica de negocios e invoca al método **Show** del objeto **IDialog** inyectado al construir la clase para que este muestre el resultado. Cada plataforma podrá implementar su propia funcionalidad en ese método.

10. Compila la solución y verifica que no haya errores.

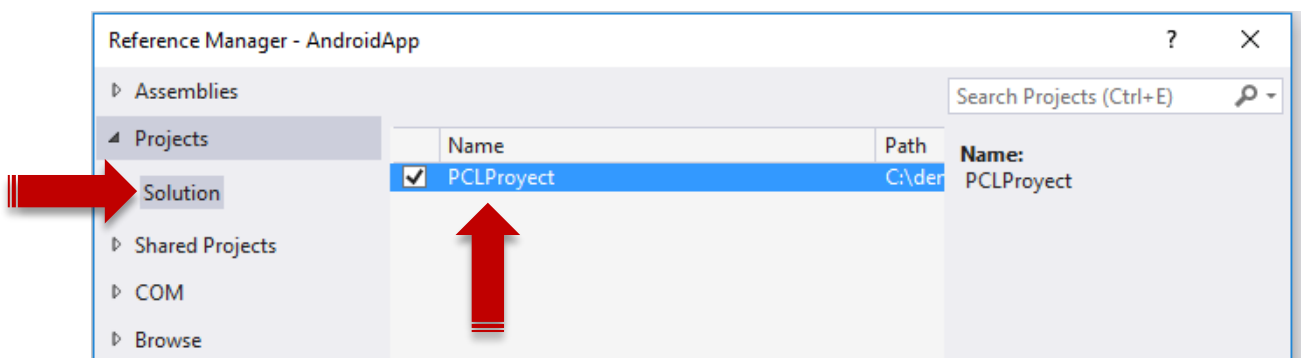


```
Output
Show output from: Build
1>----- Rebuild All started: Project: PCLProyect, Configuration: Debug Any CPU -----
1> PCLProyect -> C:\demos\Lab04\PCLProyect\bin\Debug\PCLProyect.dll
===== Rebuild All: 1 succeeded, 0 failed, 0 skipped =====
|
```

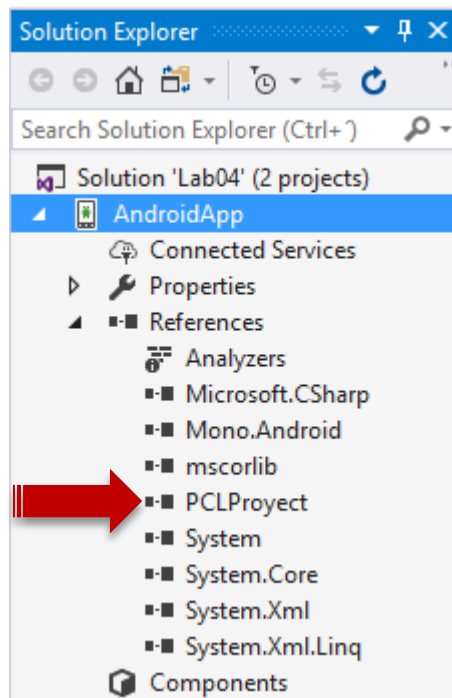
Tarea 3. Utilizando el proyecto PCL.

De la misma forma en que normalmente agregamos referencias en un proyecto, una vez que se ha creado un proyecto PCL, podemos agregar una referencia a él desde cualquier proyecto compatible de Aplicación o Biblioteca.

1. Agrega un nuevo proyecto Android a la solución utilizando la plantilla **Blank App (Android)**.
2. En el proyecto Android, agrega la referencia del proyecto PCL. La forma de agregar una referencia del proyecto PCL es similar a la forma en que se agrega una referencia a cualquier otro proyecto.

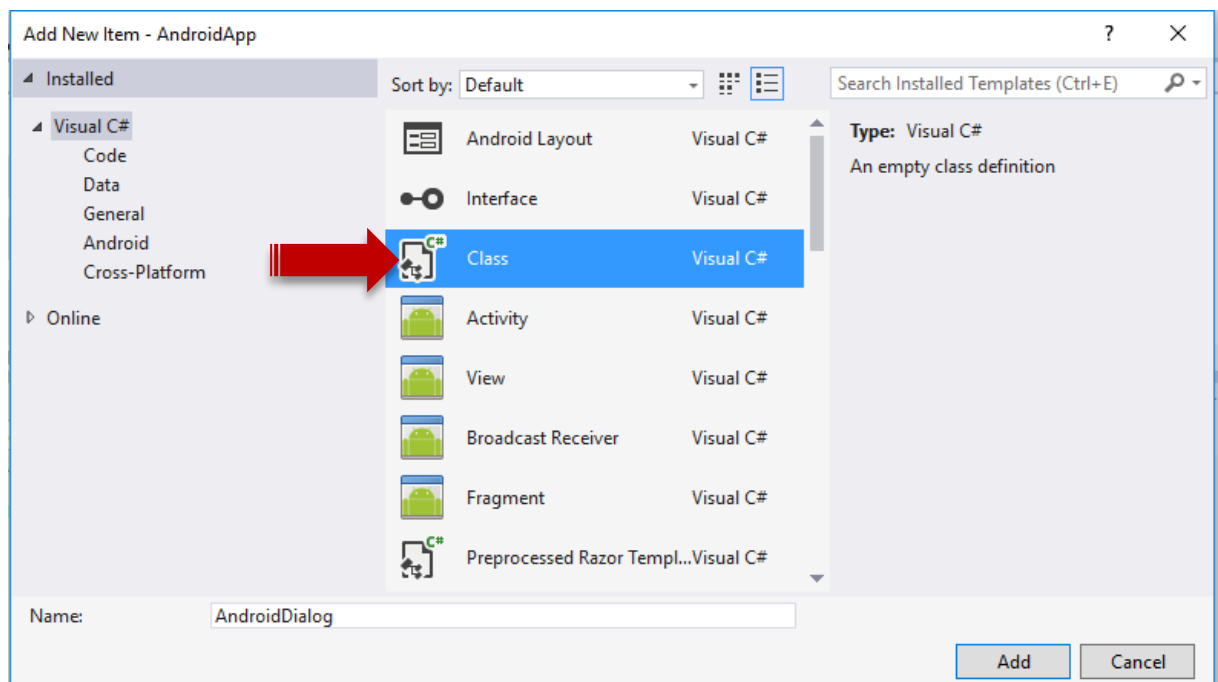


El explorador de soluciones mostrará la referencia del proyecto agregado.

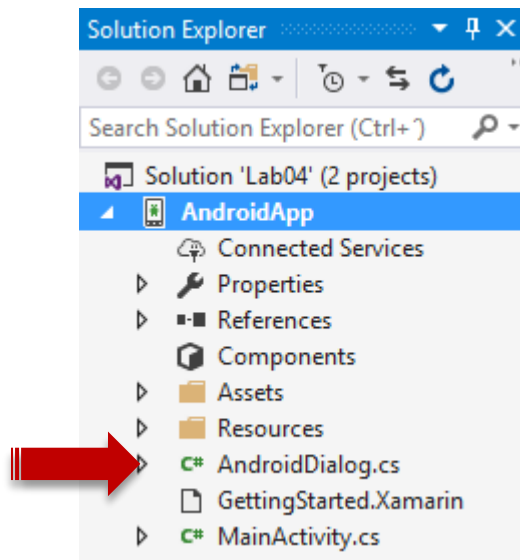


Podemos ahora implementar la funcionalidad propia de plataforma de la interface **IDialog** definida en el proyecto PCL para seguir aprovechando el código compartido.

3. Agrega un nuevo archivo de clase llamado **AndroidDialog** al proyecto Android.



El explorador de soluciones mostrará el nuevo archivo agregado.



4. Modifica la declaración de la clase **AndroidDialog** para indicar que implementa la interface **IDialog**.

```
class AndroidDialog : PCLProyect.IDialog
{
}
```

5. Agrega el siguiente código a la clase **AndroidDialog** para recibir un objeto **Context** requerido por el objeto **AlertDialog.Builder** de Android.

```
Context AppContext;
public AndroidDialog(Context context)
{
    AppContext = context;
}
```

6. Agrega el siguiente código para implementar el método **Show**. El método mostrará en pantalla el mensaje recibido utilizando clases específicas de la plataforma Android.

```
public void Show(string message)
{
    Android.App.AlertDialog.Builder Builder =
        new AlertDialog.Builder(AppContext);
    AlertDialog Alert = Builder.Create();
    Alert.SetTitle("Resultado de la verificación");
    Alert.SetIcon(Resource.Drawable.Icon);
    Alert.SetMessage(message);
    Alert.SetButton("Ok", (s, ev) => { });
    Alert.Show();
}
```

7. Abre el archivo **MainActivity.cs** del proyecto Android y modifica el método **OnCreate** para invocar al código compartido y mostrar el resultado devuelto.



```
protected override void OnCreate(Bundle bundle)
{
    base.OnCreate(bundle);

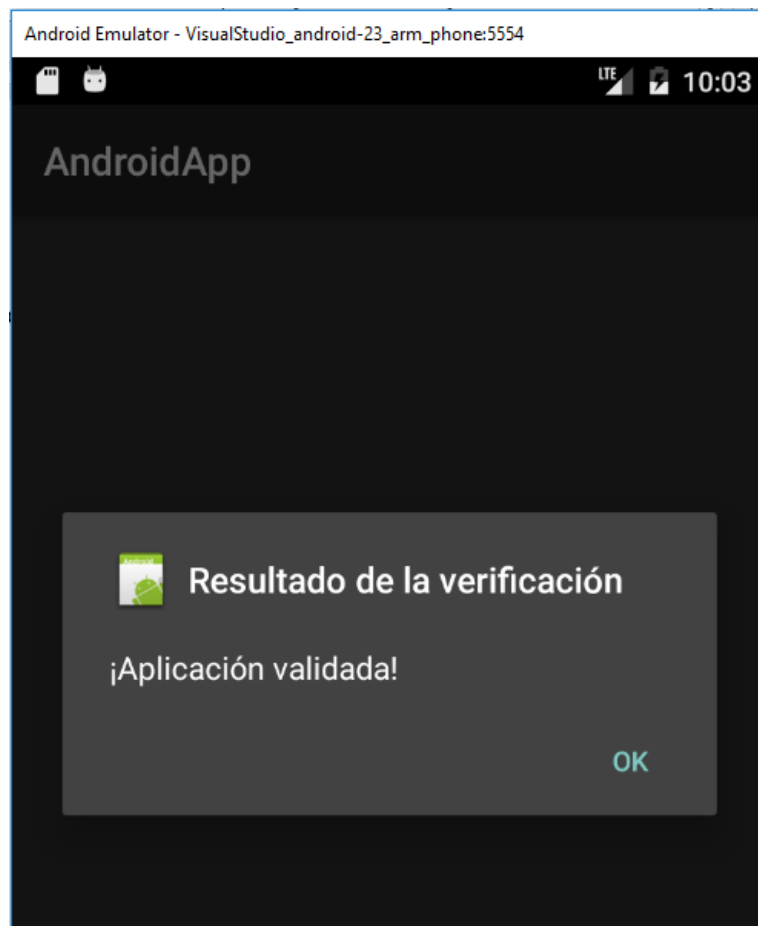
    /* Creamos la instancia del código compartido
    y le inyectamos la dependencia. */
    var Validator =
        new PCLProject.AppValidator(new AndroidDialog(this));

    /* Aquí podríamos establecer los valores de las propiedades
    * EMail, Password y Device*/

    // Realizamos la validación
    Validator.Validate();

    // Set our view from the "main" layout resource
    // SetContentView (Resource.Layout.Main);
}
```

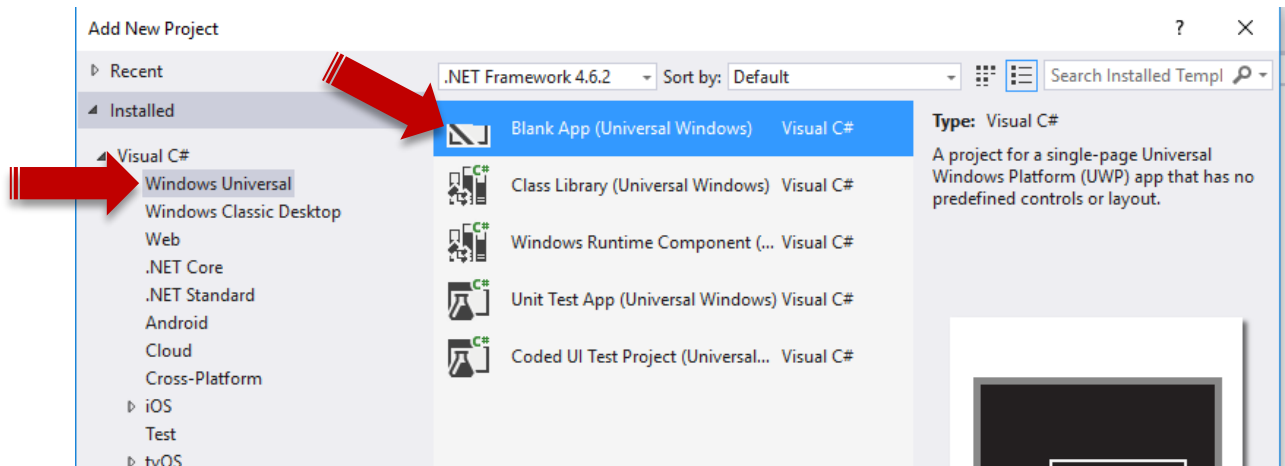
8. Establece el proyecto Android como proyecto de inicio.
9. Ejecuta la aplicación. Un mensaje será mostrado con el resultado devuelto por el método *Validate*.



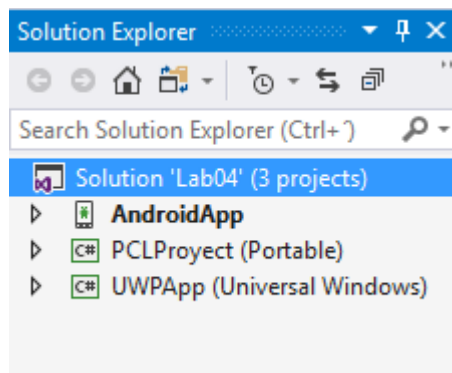


Podrás notar que el código compartido y el código específico de la plataforma se han ejecutado.

10. Agrega a la solución un nuevo proyecto para una aplicación UWP utilizando la plantilla **Blank App (Universal Windows)**.



11. El explorador de soluciones mostrará ahora los 3 proyectos.

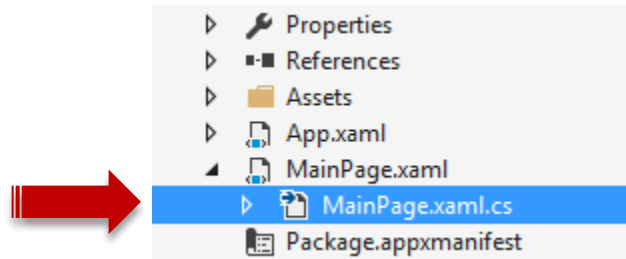


12. En el proyecto UWP, agrega la referencia del proyecto PCL.
13. Agrega un nuevo archivo de clase llamado **UWPDialog**.
14. Modifica la clase **UWPDialog** para implementar la interface **IDialog**.

```
class UWPDialog : PCLProject.IDialog
{
    public async void Show(string message)
    {
        var Dialog =
            new Windows.UI.Popups.MessageDialog(
                message);
        await Dialog.ShowAsync();
    }
}
```



15. Abre el archivo **MainPage.xaml.cs** del proyecto UWP.



16. Modifica el constructor de la clase para invocar al código compartido y mostrar el resultado devuelto.

```
public MainPage()
{
    this.InitializeComponent();

    /* Creamos la instancia del código compartido
    * y le inyectamos la dependencia. */
    var Validator =
        new PCLProject.AppValidator(
            new UWPDialog());

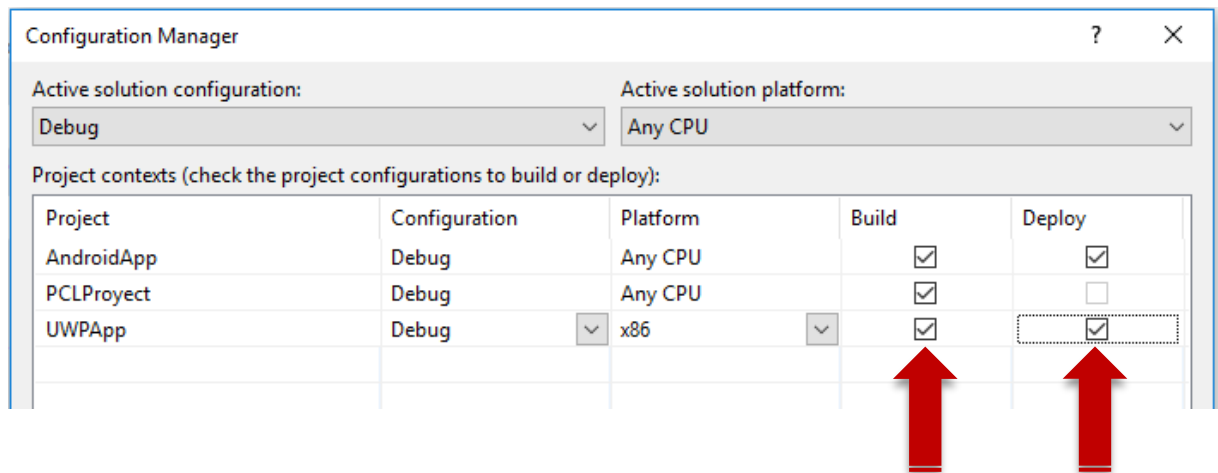
    /* Aquí podríamos establecer los valores de las propiedades
    * EMail, Password y Device*/

    // Realizamos la validación
    Validator.Validate();
}
```

17. Establece el proyecto UWP como proyecto de inicio.

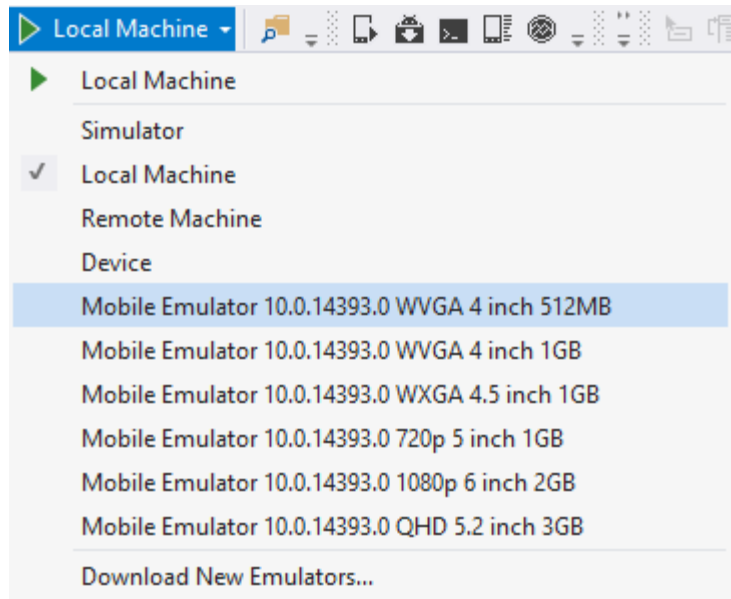
18. Accede a la opción **Configuration Manager...** del menú contextual de la solución.

19. Activa las opciones **Build** y **Deploy** del proyecto UWP y cierra la ventana **Configuration Manager**.

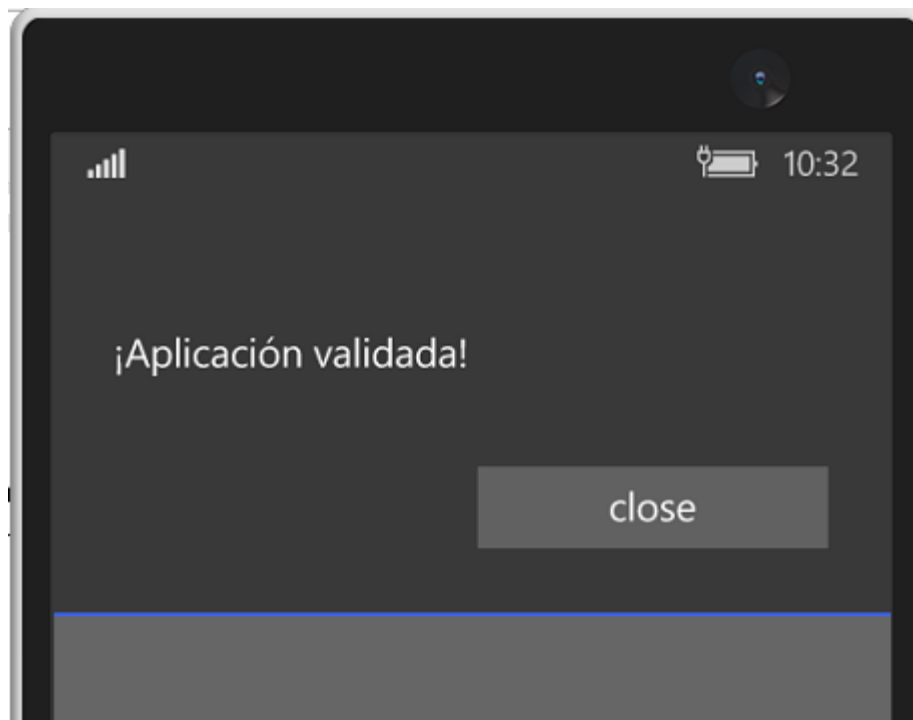




20. Selecciona un emulador donde ejecutarás tu aplicación UWP.



21. Ejecuta la aplicación. Un mensaje será mostrado con el resultado devuelto por el método *Validate*.



Podrás notar que el código compartido y el código específico de la plataforma se han ejecutado.



Ejercicio 2: Validando tu actividad

En este ejercicio agregarás funcionalidad a tu laboratorio con el único propósito de enviar una evidencia de la realización del mismo.

La funcionalidad que agregarás consumirá un ensamblado que representa una Capa de acceso a servicio (SAL) que será consumida por tu aplicación Android a través del proyecto PCL.

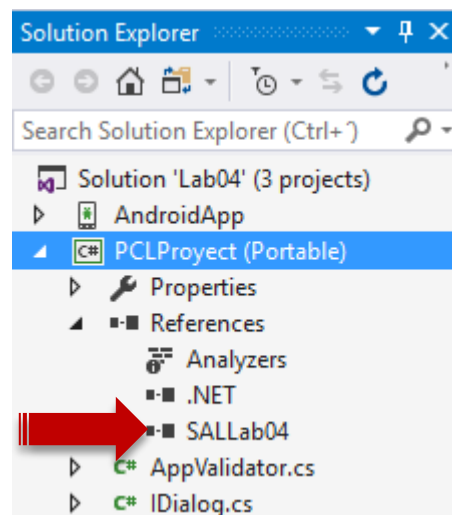
Es importante que realices cada laboratorio del diplomado ya que esto te dará derecho a obtener el diploma final del mismo.

Tarea 1. Agregar los componentes de la Capa de acceso a Servicio.

En esta tarea agregarás una referencia al ensamblado **SALLab04.dll** que implementa la capa de acceso a servicio. El archivo **SALLab04.dll** se encuentra disponible junto con este documento.

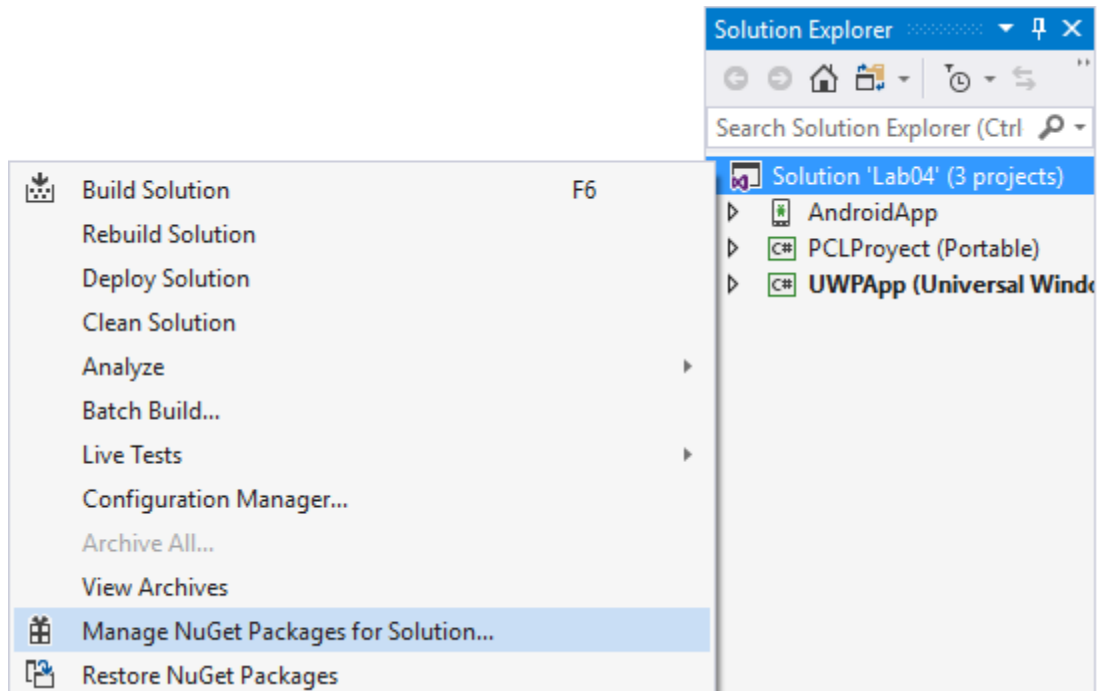
1. Accede a la opción **Add > Reference...** del menú contextual del proyecto PCL para agregar la referencia del ensamblado **SALLab04.dll**.

Puedes notar que la referencia ha sido agregada.

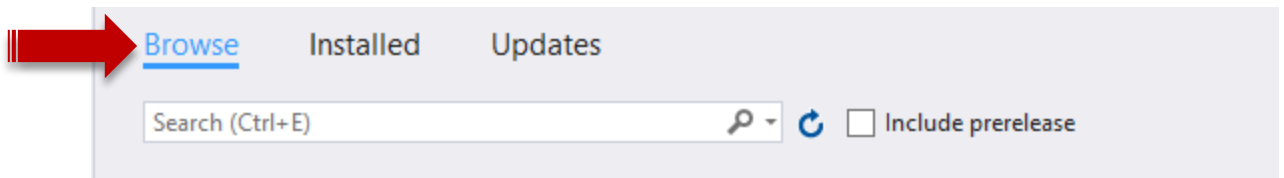


Este componente realiza una conexión a un servicio de Azure Mobile, por lo tanto, será necesario agregar el paquete NuGet **Microsoft.Azure.Mobile.Client**.

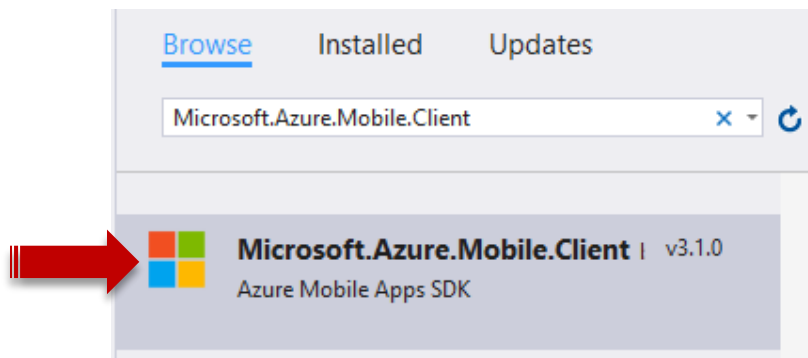
2. Accede a la opción **Manage NuGet Packages for Solution...** del menú contextual de la solución.



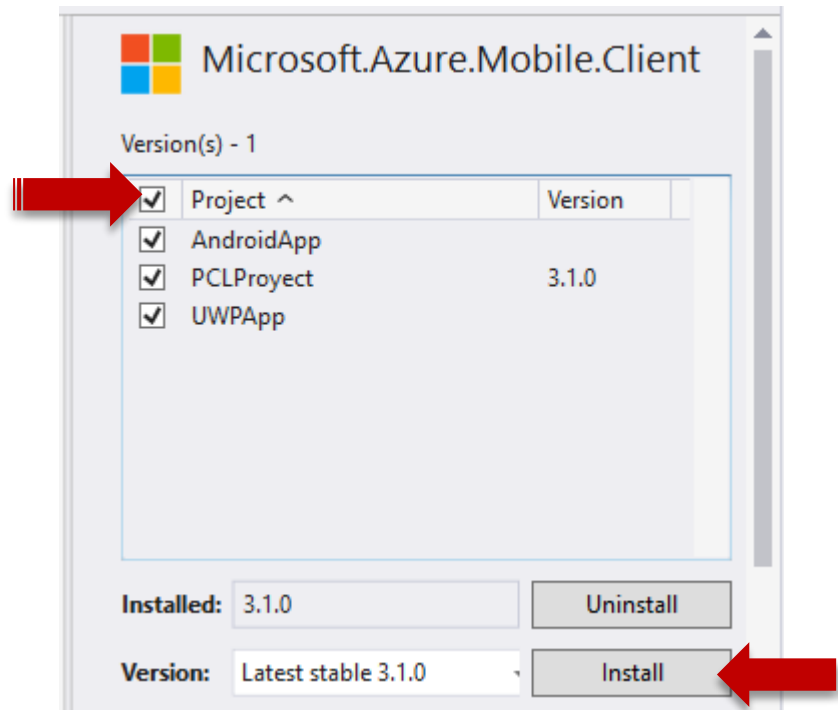
3. En la ventana **NuGet** haz clic en **Browse**.



4. En el cuadro de búsqueda escribe **Microsoft.Azure.Mobile.Client**. La lista de resultados se actualizará automáticamente conforme vas escribiendo.



5. En la ventana de resultados haz clic en **Microsoft.Azure.Mobile.Client**.
6. En el panel derecho, selecciona los 3 proyectos y haz clic en **Install** para instalar el paquete. Acepta la instalación de los paquetes adicionales cuando te sea requerido.



Tarea 2. Agregar la funcionalidad para validar la actividad.

El componente DLL que agregaste te permite registrar tu actividad en la plataforma de TI Capacitación y Microsoft. El componente se comunica con la plataforma de TI Capacitación para autenticarte y posteriormente envía un registro a la plataforma Microsoft.

1. Modifica el método `validate` del proyecto PCL para implementar la lógica de validación de la aplicación.

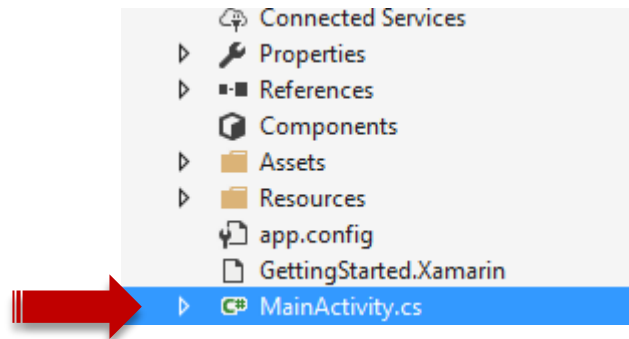
```
public async void Validate()
{
    string Result;

    var ServiceClient = new SALLab04.ServiceClient();
    var SvcResult = await ServiceClient.ValidateAsync(
        EMail, Password, Device);

    Result =
        $"{SvcResult.Status}\n{SvcResult.Fullname}\n{SvcResult.Token}";

    /* Invocar el código específico de la plataforma */
    Dialog.Show(Result);
}
```

2. Abre el archivo **MainActivity.cs** ubicado dentro del proyecto Android.



3. Agregar el siguiente código dentro del método **OnCreate**.

```
/* Aquí podríamos establecer los valores de las propiedades  
 * EMail, Password y Device*/
```

```
Validator.EMail = "TuCorreoElectrónico";  
Validator.Password = "TuContraseña";  
Validator.Device =  
    Android.Provider.Settings.Secure.GetString(  
        ContentResolver,  
        Android.Provider.Settings.Secure.AndroidId);
```

```
// Realizamos la validación  
Validator.Validate();
```

NOTA: Después de terminar el laboratorio es importante que elimines estos datos para que no queden expuestos. En laboratorios posteriores aprenderás a solicitar esos datos en tiempo de ejecución para que no queden expuestos como código duro. La comunicación del componente con la plataforma de TI Capacitación se realiza mediante HTTPS para seguridad de tu información.

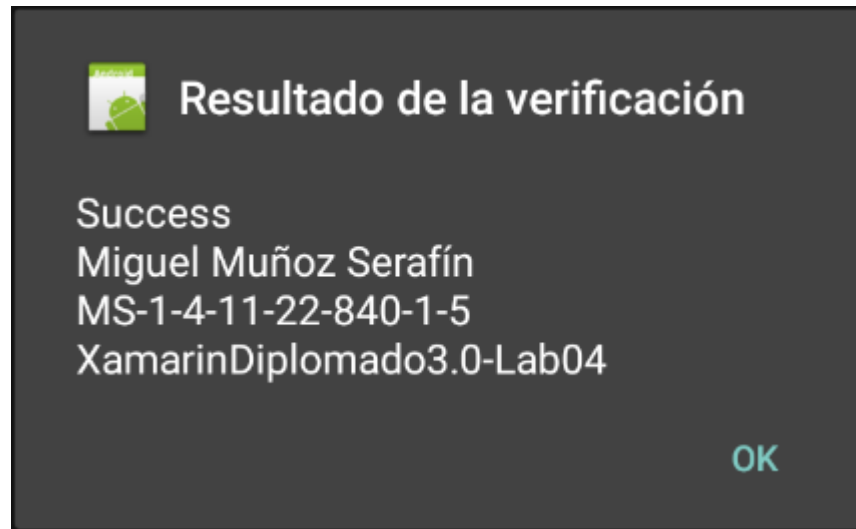
Tarea 3. Ejecutar la aplicación.

Finalmente, está todo listo para ejecutar la aplicación y realizar la validación de la actividad.

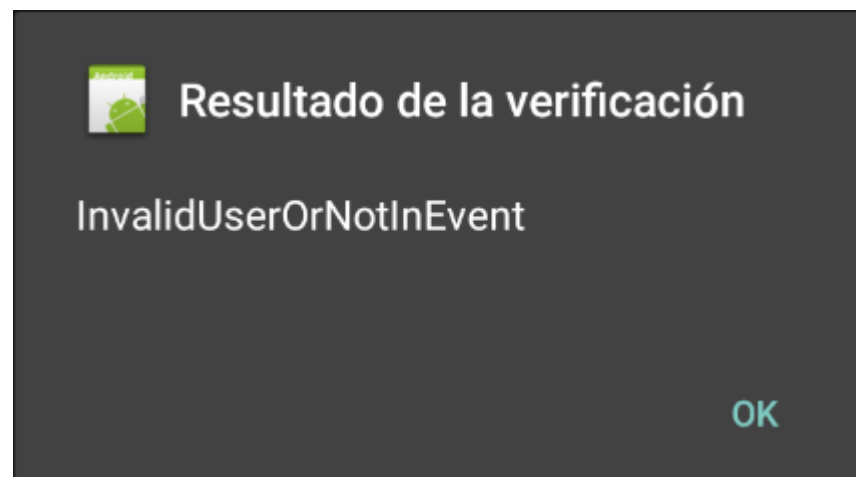
1. Selecciona la opción **Set as StartUp Project** del menú contextual del proyecto Android para establecerlo como proyecto de inicio.
2. Selecciona el emulador Android de tu preferencia y ejecuta la aplicación.

La aplicación será ejecutada en el emulador e iniciará el proceso de validación de la actividad. Al finalizar la validación se mostrará un cuadro de diálogo mostrando tus datos y el estatus de la validación.

Si la validación se realiza con éxito, te será mostrado tu nombre completo y el estatus **Success** como en la siguiente imagen.



Si proporcionaste tus credenciales incorrectas, te mostrará un mensaje similar al siguiente.



Cuando tu actividad se haya validado exitosamente puedes ver el estatus en el siguiente enlace: <https://ticapacitacion.com/evidencias/xamarin30>.

3. Regresa a Visual Studio y detén la ejecución de la aplicación.
4. **Elimina los datos de tus credenciales de acceso a la plataforma de TI Capacitación.**

Nota: Es probable que recibas un correo similar al siguiente.

Tu código de lab no es válido, revisa que estés utilizando un código de reto válido. Si tienes preguntas o dudas por favor contacta a dxaudmx@microsoft.com

Puedes hacer caso omiso al mensaje.



Si encuentras problemas durante la realización de este laboratorio, puedes solicitar apoyo en los grupos de Facebook siguientes:

<https://www.facebook.com/groups/iniciandoconxamarin/>

<https://www.facebook.com/groups/xamarindiplomadoitc/>



Resumen

En este laboratorio exploraste la forma en que trabajan los proyectos de Bibliotecas de Clases Portables comúnmente conocidos como **Portable Class Libraries o PCL**.

Una tercera opción para compartir código es mediante bibliotecas **.NET Standard** que representan una evolución de las bibliotecas PCL. La manera de utilizarlas es similar a lo que aprendiste en este laboratorio, la única diferencia es que para trabajar con .NET Standard debes seleccionar la plantilla **Class Library (.NET Standard)** en lugar de la plantilla **Class Library (Portable)**. El enfoque del uso de interfaces y clases abstractas para compartir código es el mismo.

¿Qué te pareció este laboratorio?

Comparte tus comentarios en twitter y Facebook utilizando el hashtag **#XamarinDiplomado**.