

Publicado por
Microsoft Press
Una división de Microsoft Corporation One
Microsoft Way
Redmond, Washington 98052-6399

Copyright © 2016 Xamarin, Inc.

Todos los derechos reservados. Ninguna parte del contenido de este libro puede ser reproducida o transmitida en cualquier forma o por cualquier medio sin el permiso por escrito del editor.

ISBN: 978-1-5093-0297-0

Impreso y encuadrado en los Estados Unidos de América.

primera impresión

Microsoft Press libros están disponibles a través de libreros y distribuidores en todo el mundo. Si necesita apoyo relacionados con este libro, el correo electrónico de Microsoft Press Support en mspinput@microsoft.com. Por favor, díganos lo que piensa de este libro en <http://aka.ms/tellpress>.

Este libro se proporciona "tal cual" y expresa puntos de vista y las opiniones del autor. Los puntos de vista, opiniones e informaciones expresadas en este libro, incluidas las direcciones URL y otras referencias a sitios web de Internet, pueden cambiar sin previo aviso.

Algunos ejemplos descritos en el presente documento se proporcionan solamente con ilustración y son ficticios. No se encontró asociación o conexión real se pretende ni se debe inferir.

Microsoft y las marcas que figuran en <http://www.microsoft.com> en la página web "marcas" son marcas comerciales del grupo de compañías de Microsoft. Todas las demás marcas son propiedad de sus respectivos dueños.

Adquisiciones y Project Editor: Devon Musgrave

Producción editorial: John Pierce, ardilla voladora Press

Ilustración de la cubierta: Serena Zhang

Contenido

<i>Prólogo</i>	xv
<i>Introducción</i>	xvi
Capítulo 1 ¿Cómo Xamarin.Forms encajar?	1
Cruz-plataforma de desarrollo móvil	2
El panorama móvil	2
Problema 1: diferentes paradigmas de interfaz de usuario	2
Problema 2: Los diferentes entornos de desarrollo	3
Problema 3: Diferentes interfaces de programación de	3
Problema 4: Diferentes lenguajes de programación	3
El C # y solución .NET	4
Un solo idioma para todas las plataformas	5
Compartir código	6
La introducción de Xamarin.Forms	8
La opción Xamarin.Forms	8
apoyo XAML	13
especificidad plataforma	14
Una panacea multiplataforma?	15
Su entorno de desarrollo	15
Máquinas y entornos de desarrollo	dieciséis
Dispositivos y emuladores	dieciséis
Instalación	17
La creación de una aplicación para iOS	17
La creación de una aplicación para Android	18
Creación de una aplicación de Windows	18
¿Listo?.....	19

Capítulo 2 Anatomía de una aplicación	20
Di hola	20
Dentro de los archivos	24
El proyecto IOS	26
El proyecto Android	27
El proyecto Plataforma Windows universal	28
¡Nada especial!	28
PCL o SAP?	29
Etiquetas para el texto	31
Solución 1. Incluir el relleno en la página	35
Solución 2. Incluir el relleno sólo para iOS (SAP solamente)	36
Solución 3. Incluir el relleno sólo para iOS (PCL o SAP)	37
Solución 4. Centro de la etiqueta dentro de la página	39
Solución 5. Centrar el texto dentro de la etiqueta	41
Capítulo 3 Más profundamente en el texto	42
párrafos envolver	42
Texto y colores de fondo	44
La estructura de color	46
Cambiar el esquema de color de la aplicación	50
tamaños y atributos de fuente	51
El texto con formato	53
Capítulo 4 hojear en la pila	60
Las pilas de vistas	60
El desplazamiento del contenido	64
La opción expande	70
Marco y BoxView	74
Un ScrollView en un StackLayout?	82
Capítulo 5 Manejo de tamaños	87
Píxeles, puntos, dps, salsa, y UDE	87
tamaños métricos	96

tamaño de las fuentes estimadas	97
ajuste de texto disponible a	99
Un reloj de ajuste a tamaño	103
Los problemas de accesibilidad	105
texto Empíricamente apropiado	108
Capítulo 6 clics del botón	113
El procesamiento de la clic	113
Compartir pulsaciones de botón	116
controladores de eventos anónimos	119
Distinguir puntos de vista con los ID	121
Guardar los datos transitorios	124
Capítulo 7 XAML código vs.	131
Propiedades y atributos	132
sintaxis de la propiedad de elementos	136
Añadir una página XAML a su proyecto	140
El compilador XAML	145
especificidad plataforma en el archivo XAML	146
El atributo de propiedad de contenido	150
El texto con formato	152
Capítulo 8 código y XAML en armonía	156
Pasar argumentos	156
Constructores con argumentos	156
¿Puedo llamar a los métodos de XAML?	159
El atributo x: Nombre	161
Las vistas personalizadas basadas en XAML	167
Eventos y manejadores	172
Toque gestos	175
El capítulo 9 de plataforma específica llamadas a la API	182
Preprocesamiento en el Proyecto activo compartido	182
clases paralelas y el Proyecto activo compartido	185

DependencyService y la biblioteca de clases portátil	187
la generación de sonido específico de la plataforma	191
Capítulo 10 extensiones de marcado XAML	198
La infraestructura de código	198
El acceso a miembros estáticos	200
diccionarios de recursos	206
StaticResource para la mayoría de los propósitos	207
Un árbol de diccionarios	214
DynamicResource para fines especiales	218
Menos utilizadas extensiones de marcado	221
Una extensión de marcado personalizada	222
Capítulo 11 La infraestructura enlazable	227
La jerarquía de clases Xamarin.Forms	228
Una ojeada en bindableObject y BindableObject	235
Definir las propiedades enlazables	241
La propiedad enlazable de sólo lectura	246
Capítulo 12 estilos	252
El estilo básico	252
Estilos de código	259
herencia de estilos	261
estilos implícitos	265
estilos dinámicos	270
estilos de dispositivos	278
Capítulo 13 Los mapas de bits	283
mapas de bits independientes de la plataforma	284
Montar y llenar	287
recursos incrustados	289
Más sobre dimensionamiento	295
Navegar y esperando	307
mapas de bits de transmisión	311

Acceder a los flujos	311
La generación de mapas de bits en tiempo de ejecución	314
mapas de bits específicos de la plataforma	318
resoluciones de mapa de bits	320
mapas de bits independientes del dispositivo para iOS	322
mapas de bits independientes del dispositivo para Android	322
mapas de bits independientes del dispositivo de tiempo de ejecución para las plataformas Windows	323
Barras de herramientas y sus iconos	327
Iconos para Android	329
Iconos para las plataformas de Windows Runtime	330
Los iconos de los dispositivos IOS	331
imágenes de los botones	335
Capítulo 14 de disposición absoluta	338
AbsoluteLayout en el código	339
Adjunta propiedades enlazables	344
dimensionamiento y posicionamiento proporcional	348
Trabajar con coordenadas proporcionales	350
AbsoluteLayout y XAML	355 ..
Superposiciones	359
Un poco de diversión	362
Capítulo 15 La interfaz interactiva	371
Ver descripción general	371
Deslizador y paso a paso	372
fundamentos deslizante	372
Errores comunes	375
selección de color deslizador	377
La diferencia de pasos	382
Cambiar y CheckBox	384
conceptos básicos de conmutación	384
Un CheckBox tradicional	387

Escribir el texto	392
Teclado y el enfoque	392
La elección del teclado	393
Propiedades de entrada y eventos	396
La diferencia Editor	402
El SearchBar	405
Fecha y selección de tiempo	410
DatePicker	410
El TimePicker (o es una TimeSpanPicker?)	414
vinculante Capítulo 16 Datos	418
fundamentos vinculantes	418
Código y XAML	420
Fuente y BindingContext	423
El modo de unión	430
El formato de cadenas	437
Por qué se llama "Camino"?	440
Encuadernación convertidores de valores	443
Fijaciones y vistas personalizadas	451
Capítulo 17 El dominio de la cuadrícula	458
La red básica	458
La cuadrícula en XAML	458
La cuadrícula en el código	464
El gráfico de cuadrícula bar	467
La alineación en la cuadrícula	471
divisores celulares y fronteras	475 ..
Casi ejemplos de cuadrícula de la vida real	481
En respuesta a los cambios de orientación	484
Capítulo 18 MVVM	491
interrelaciones MVVM	491
ViewModels y el enlace de datos	493

Un reloj modelo de vista	494
propiedades interactivas en un ViewModel	500
Un modelo de vista del color	507
La racionalización del modelo de vista	513
La interfaz de comandos	517
ejecuciones método simple	519
Una calculadora, casi	523
ViewModels y el ciclo de vida de aplicaciones	531
Capítulo 19 vistas Collection	535
Las opciones del programa con el Selector de	536
El Selector de control de eventos y	536
El enlace de datos del Selector de	540
La representación de datos con ListView	543
Colecciones y selecciones	544
El separador de fila	545
El enlace de datos del elemento seleccionado	547
La diferencia ObservableCollection	551
Plantillas y células	553
células personalizados	561
La agrupación de los elementos de ListView	565
encabezados de grupo personalizada	569
ListView y la interactividad	570
ListView y MVVM	574
Una colección de ViewModels	574
La selección y el contexto de unión	586
Los menús contextuales	590
La variación de los elementos visuales	595
Refrescante el contenido	597
El TableView y sus intentos	604
Propiedades y jerarquías	604

Una forma prosaica	606
células personalizados	610
secciones condicionales	616
Un menú TableView	620
Capítulo 20 asincrónico y el archivo de E / S	624
A partir de las devoluciones de llamada en espera	625
Una alerta con las devoluciones de llamada	626
Una alerta con lambdas	630
Una alerta con lo esperan	630
Una alerta sin nada	632
Guardar los ajustes del programa de forma asíncrona	635
Un temporizador independiente de la plataforma	639
Archivo de entrada / salida	641
Buenas y malas noticias	641
Un primer tiro en archivos de plataforma cruzada que	643
complacientes archivo de Windows en tiempo de ejecución de E / S	650
bibliotecas específicas de la plataforma	651
Manteniéndolo en el fondo	662
No bloquee el hilo de interfaz de usuario!	664
Sus propios métodos awaitable	664
El Mandelbrot básico Da	666
Que marca un avance	673
Cancelación del trabajo	676
Un Mandelbrot MVVM	680
Volver a la web	701
Capítulo 21 Transformaciones	704
La traducción transformar	705
Efectos de texto	708
Saltos y animaciones	711
transformar la escala	715

El anclaje de la escala	723
La rotación transformar	726
efectos de texto girado	727
Un reloj analógico	734
deslizadores verticales?	742
rotaciones 3D-ish	743
Capítulo 22 Animación	748
Explorar animaciones básicas	748 ..
Ajuste de la duración de la animación	750
animaciones relativas	751
En espera de animaciones	751
animaciones compuestas	754
Task.WhenAll y Task.WhenAny	756
Rotación y anclajes	756
Funciones de aceleración	759
Sus propias funciones de aceleración	763
animaciones de entrada	773
Siempre animaciones	778
La animación de la propiedad límites	793
Sus propias animaciones awaitable	805
Más profundamente en la animación	808
La clasificación de las clases	808
clase ViewExtensions	809
La clase de Animación	810
AnimationExtensions clase	811
Trabajar con la clase de Animación	812
animaciones infantiles	815
Más allá de los métodos de animación de alto nivel	818
Más de sus propios métodos awaitable	819
La implementación de una animación de Bezier	823

Trabajar con AnimationExtensions	828
La estructuración de sus animaciones	834
Capítulo 23 disparadores y comportamientos	835
Disparadores	836
El gatillo simple	836
Desencadenar acciones y animaciones	841
Más evento activa	847
Datos desencadena	853
La combinación de las condiciones en el MultiTrigger	858
Comportamientos	868
Comportamientos con propiedades	871
Permite cambiar las cajas de verificación y	876 ..
Responder a los toques	889
Botones de radio	893
Fundidos y orientación	908
Capítulo 24 páginas La navegación	920
páginas modales y no modales páginas	921
transiciones de página animadas	930
variaciones visuales y funcionales	931
Explorar la mecánica	937 ..
Hacer cumplir modalidad	947
variaciones de navegación	955
Hacer un menú de navegación	957
La manipulación de la pila de navegación	963
generación de páginas dinámicas	965
Los patrones de transferencia de datos	969
argumentos de constructor	970
Propiedades y las llamadas a métodos	974
El centro de mensajería	981
Eventos	984

El intermediario clase de aplicaciones	987
El cambio a un modelo de vista	990
Guardado y restablecimiento de estado de la página	995
Guardado y restablecimiento de la pila de navegación	999
Algo así como una aplicación en la vida real	1006
Capítulo 25 variedades página	1020
Maestra y detalle	1020
La exploración de los comportamientos	1021
De vuelta a la escuela	1029
Su propia interfaz de usuario	1034 ..
TabbedPane	1042
páginas de fichas discretos	1043
El uso de un ItemTemplate	1050 ..
Capítulo 26 diseños personalizados	1054
Una visión general de diseño	1054
Padres e hijos	1055
Tamaño y posicionamiento	1056 ..
Restricciones y solicitudes de tamaño	1060
limitaciones infinitas	1064
El mirar a escondidas dentro del proceso	1066
Derivado de diseño <Ver>	1074
Un ejemplo fácil	1075
posicionamiento vertical y horizontal simplifica	1082
La invalidación	1084
Algunas reglas para los diseños de codificación	1087
Un diseño con propiedades	1088
No hay dimensiones sin restricciones permitidas	1099
La superposición de los niños	1106
Más adjunto propiedades enlazables	1117
Diseño y LayoutTo	1122

Capítulo 27 procesadores personalizados	1127
La jerarquía completa de clase	1127
Hola, a medida renderizadores!	1130
Renderizadores y propiedades	1135 ..
Extracción de grasas y eventos	1149

Prefacio

La idea de producir un libro sobre Xamarin.Forms es uno que hemos tenido durante casi tanto tiempo como hemos estado trabajando en el producto. Por supuesto, no sabíamos que sería escrita por un autor tan talentoso y de gran prestigio. No podríamos haber pedido una mejor persona calificada, ni alguien que requeriría tan poco de nosotros para entrar en nuestra mente! Charles ofrece una visión de una manera tan bellas y simples, como pronto descubrirá.

Este libro destila más de tres años de esfuerzo para crear un moderno, kit de herramientas multiplataforma como fácil de entender, la progresión de ideas organizada. Los ejemplos que aparecen en este libro son lo suficientemente simple para ser entendido sin la necesidad de un IDE de fantasía o compilador, aunque mantienen la complejidad requerida para ser aplicable a los problemas que enfrentan las aplicaciones reales.

Mejor, los siguientes capítulos no se centran en una sola plataforma, pero tienen un enfoque holístico para entender el desarrollo móvil para todas las plataformas, no sólo para iOS o Android o Windows.

Hemos querido evitar los peligros asociados comúnmente con kits de herramientas multiplataforma: o bien tienen una experiencia de usuario alien-sentimiento, o que se limitan a un mínimo común denominador en todas las plataformas de destino. El patrón que se enamoró era utilizar las API nativas, como es la manera tradicional Xamarin. Xamarin.Forms ofrece al usuario el subconjunto más pequeño utilizable de APIs que se requieren para escribir la mayor parte de una aplicación en una base de código unificado, y luego da acceso a la caja de herramientas subyacente de ajuste y acabado. El resultado final es que el usuario tiene la capacidad de expresar la mayoría de su aplicación en código unificado, sin perder la flexibilidad de aplicación por cada plataforma.

Funciona, también, mediante la eliminación de la necesidad de proporcionar todas las funciones dentro de la abstracción. En su lugar, permitimos que el simple acceso a la caja de herramientas para que los desarrolladores de aplicaciones son capaces de llevar a cabo esas funciones platformspecific que hacen brillar su aplicación. El noventa por ciento de lo que hace su trabajo aplicación es el mismo que para cualquier otro aplicación por ahí, pero trabajando a través de plataformas no debe obligar a renunciar al 10 por ciento que hace que su aplicación única.

Debido a esto, Xamarin.Forms es en muchos sentidos el "untoolkit", un conjunto de herramientas que no es tanto un conjunto de herramientas, ya que es una manera de ver el desarrollo móvil y utilizarlo como un patrón para crear aplicaciones para móviles. Si los autores de Xamarin.Forms le puede ofrecer nada para retener a medida que lee este libro, es que los kits de herramientas, plataformas y tecnologías cambian muy rápidamente, pero los patrones, especialmente buenos patrones, rara vez mueren.

Cuando leí las ediciones previas de este libro, me voló la cabeza. Charles entiende lo que estábamos tratando de hacer mejor que nadie ha tenido. Este libro está escrito a sabiendas de que se trata de Xamarin.Forms el patrón de la creación de aplicaciones para móviles. Creo que por el momento en que termine la lectura, usted también va a entender qué es lo que propusimos crear.

Xamarin.Forms co-creador, Jason
Smith

Introducción

Esta es la tercera versión de un libro sobre la escritura de aplicaciones con Xamarin.Forms, la plataforma de desarrollo móvil emocionante para iOS, Android y Windows dado a conocer por Xamarin en mayo de 2014. (Las dos primeras versiones de este libro fuera de vista previa ediciones.) Xamarin.Forms le permite escribir código de usuario-interfaz compartida en C# y XAML (Lenguaje de marcado Extensible Application) que se asigna a los controles nativos en estas plataformas.

El soporte de Windows de Xamarin.Forms incluye Windows Runtime (WinRT) para la orientación de Windows 8.1 y dispositivos Windows Phone 8.1, y la plataforma Windows universal (UWP), que es una forma del tiempo de ejecución de Windows que se dirige a Windows 10 y Windows 10 Los dispositivos móviles con un solo programa.

Las dos versiones anteriores de este libro fueron llamados vista previa ediciones, ya que no estaban completos. En 1200 páginas, esta es la primera edición que se pretende ser completa, a pesar de que varios temas que no están incluidos y Xamarin.Forms se sigue perfeccionando progresivamente sin signos de desaceleración.

Toda la información acerca de este libro se puede encontrar en la página principal del libro en:

<https://developer.xamarin.com/r/xamarin-forms/book/>

Quién debería leer este libro

Este libro es para programadores de C# que quieren escribir aplicaciones utilizando una única base de código que se dirige a las tres plataformas móviles más populares: iOS, Android y Windows, que abarca la plataforma Windows de teléfono universal y Windows.

Xamarin.Forms también tiene aplicabilidad para aquellos programadores que eventualmente desean utilizar C# y las bibliotecas Xamarin.iOS y Xamarin.Android apuntar a las interfaces de programación de aplicaciones (API) nativas de estas plataformas. Xamarin.Forms pueden ser de gran ayuda en la obtención de los programadores comenzando con estas plataformas o en la construcción de un prototipo o una aplicación de prueba de concepto.

Este libro supone que sabe C# y está familiarizado con el uso de .NET Framework. Sin embargo, cuando discuto algunas características de C# y .NET que podrían ser un tanto exótico o desconocido para los últimos programadores de C#, adopto un ritmo algo más lento.

Convenios y características en este libro

Este libro tiene sólo unas pocas convenciones tipográficas:

- Todos los elementos de programación que se hace referencia en el texto, incluyendo las clases, métodos, propiedades, nombres de variables, etc.-se muestran en una fuente monoespaciada, tales como la StackLayout clase.
- Los artículos que aparecen en la interfaz de usuario de Visual Studio o Xamarin de estudio, o las aplicaciones discutidas en estos capítulos, aparecen en negrita, como el **Agregar nuevo proyecto** diálogo.
- soluciones y proyectos de aplicación también aparecen en negrita, como **MonkeyTap**.

Las distintas ediciones de este libro

Este libro pretende ser un tutorial para aprender a programar en Xamarin.Forms. No es un sustituto de la documentación de la API, que se puede encontrar en el enlace marco Xamarin.Forms en esta página:

<https://developer.xamarin.com/api/>

El primer adelanto edición de este libro fue publicado en octubre de 2014 para coincidir con la conferencia Xamarin Evolve 2014. Contenía seis capítulos, pero no hay cobertura de XAML.

Esta segunda edición de vista previa se reconcebida para contener capítulos más cortos y más específicos. Los dieciséis capítulos de la segunda Edición preliminar se publicaron en abril de 2015, coincidiendo con la conferencia de Microsoft Build 2015. Durante los siguientes seis meses, ocho más capítulos fueron publicados en línea, llevando el total a 24.

Esta edición cuenta con 27 capítulos y está siendo publicado para coincidir con el Xamarin Evolve conferencia de 2016 que tendrá lugar 24-28 de abril de 2016. Sin embargo, la fecha límite para este libro es alrededor de un mes antes de lo Evolve, y varios temas que no lo hacen en esta edición . Estos incluyen mapas, ControlTemplate,

DataTemplateSelector, el Margen propiedad, y CarouselView. De las clases que se derivan de GestureRecognizer, solamente TapGestureRecognizer está cubierto, y no PanGestureRecognizer o PinchGestureRecognizer. A pesar de que Disposición relativa data de la primera liberación de Xamarin.Forms, de alguna manera, nunca lo hizo en este libro.

Entre la segunda Edición de Vista previa y esta edición, se produjo un gran cambio para las plataformas Windows: Los programas de ejemplo ya no apoyan la API de Silverlight de Windows Phone 8.0. En su lugar, todos los programas de ejemplo soportan la plataforma Windows universal para la orientación de Windows 10 y Windows 10 móvil, y el tiempo de ejecución de Windows para la orientación de Windows 8.1 y Windows Phone 8.1.

Sin embargo, no hubo tiempo suficiente para actualizar los programas y capturas de pantalla de ejemplo de este libro para reflejar Android AppCompat y materiales de diseño, que se espera que esté apoyado en un próximo Visual Studio y plantilla de proyecto Xamarin Estudio de Xamarin.Forms.

Para las actualizaciones y adiciones a esta edición, compruebe la página web Xamarin dedicado a este libro.

Requisitos del sistema

Este libro supone que va a utilizar Xamarin.Forms para escribir aplicaciones que se dirigen simultáneamente todas las plataformas de iOS móviles compatibles, Android, la plataforma Windows universal, y tal vez Windows Phone 8.1 también. Sin embargo, es posible que algunos lectores se dirigen sólo una o dos plataformas en sus soluciones Xamarin.Forms. Las plataformas de las que oriente gobiernan los requisitos de hardware y software. Para dirigir los dispositivos iOS, se necesita un Mac de Apple Xcode instalado con la Plataforma y Xamarin, que incluye Xamarin Studio. Para dirigir cualquiera de las plataformas de Windows, necesitará Visual Studio 2015 en un PC, y tendrá que tener instalado la Plataforma Xamarin.

Sin embargo, también se puede utilizar Visual Studio en el PC a los dispositivos de iOS a través de una conexión Wi-Fi accesible Mac instalado con Xcode y la Plataforma Xamarin. Puede orientar los dispositivos Android de Visual Studio en el PC o desde Xamarin Estudio en el Mac.

Capítulo 1, “¿Cómo encajan en Xamarin.Forms?” Tiene más detalles sobre las diferentes configuraciones que puede utilizar y recursos para obtener información adicional y apoyo. Mi disposición para la creación de este libro consistió en una Microsoft Surface Pro 2 (con monitor externo, teclado y ratón) instalado con Visual Studio 2015 y la Plataforma Xamarin, conectados por Wi-Fi con un MacBook Pro instalado con Xcode y la Plataforma Xamarin.

La mayor parte de las capturas de pantalla en este libro muestran un iPhone, un teléfono Android, y un dispositivo móvil de Windows 10 en ese orden. Los tres dispositivos mostrados en estas imágenes reflejan mi configuración y el hardware:

- El simulador de iPhone 6 en el IOS de funcionamiento Pro MacBook 8.2
- Una 6.0.1 Android LG Nexus 5 corriendo
- Un Nokia Lumia 925 funcionando 10 en Windows Mobile

Las capturas adicionales utilizan un simulador de Aire 2 IPAD, un Microsoft Surface Pro 3 con Windows 10 en el modo de tableta, un teléfono móvil de Windows 10 que ejecuta un programa de orientación de Windows Phone 8.1, y el escritorio de Windows 10 que ejecuta un programa de orientación de Windows 8.1.

Algunas de las primeras imágenes de triples en este libro utilizan dispositivos con versiones anteriores tanto de los sistemas operativos, por ejemplo Android 5.0 o 5.1. Aunque traté de usar dispositivos reales para todas las capturas de pantalla Android y Windows, en aras de la conveniencia alguna Windows Phone y Windows Mobile 10 capturas de pantalla fueron tomadas con un emulador de Windows Mobile 10.

Descargas: Ejemplos de código

Los programas de ejemplo que se muestran en las páginas de este libro fueron compilados a finales de marzo 2016 Xamarin.Forms versión 2.1.0. El código fuente de estas muestras está alojado en un repositorio en GitHub:

<http://aka.ms/xamarinbook/codesamples>

Puede clonar la estructura de directorios en una unidad local en su máquina o descargar una gran carpeta ZIP. Voy a tratar de mantener el código actualizado con la última versión de Xamarin.Forms y fijar (y comentario) cualquier error que pueda haber colado a través.

Puede informar de los problemas, errores u otro tipo de retroalimentación sobre el código fuente de libro o haciendo clic en el **Cuestiones botón en esta página GitHub.** Puede buscar a través de los problemas existentes o presentar una nueva. Para presentar una nueva edición, tendrá que unirse a GitHub (si no lo ha hecho).

Utilice esta página GitHub sólo para cuestiones relacionadas con el libro. Para preguntas o discusiones acerca de Xamarin.Forms si mismo, utilizar el foro Xamarin.Forms:

<http://forums.xamarin.com/categories/xamarin-forms>

La actualización de los ejemplos de código

Las bibliotecas que componen Xamarin.Forms se distribuyen a través del gestor de paquetes NuGet. El paquete Xamarin.Forms consiste en una colección de bibliotecas de vínculos dinámicos, siendo la más significativa de las cuales son:

- Xamarin.Forms.Core.dll
- Xamarin.Forms.Xaml.dll
- Xamarin.Forms.Platform.dll
- Xamarin.Forms.Platform.iOS.dll
- Xamarin.Forms.Platform.Android.dll
- Xamarin.Forms.Platform.WinRT.dll
- Xamarin.Forms.Platform.WinRT.Phone.dll
- Xamarin.Forms.Platform.WinRT.Tablet.dll
- Xamarin.Forms.Platform.UAP.dll

El paquete Xamarin.Forms también requiere cinco bibliotecas de soporte para Android, actualmente identificados con el número de versión 23.0.1.3. Estos deben ser incluidos automáticamente.

Cuando se crea una nueva solución Xamarin.Forms utilizando Visual Studio o Xamarin Studio, una versión del paquete Xamarin.Forms se convierte en parte de esa solución. Sin embargo, eso podría no ser la última versión disponible de Xamarin.Forms NuGet. Es probable que desee actualizar ese paquete a la versión más reciente.

Además, el código fuente de este libro que se almacena en GitHub no incluye los paquetes reales NuGet. Xamarin Estudio descargará automáticamente cuando se carga la solución, pero por defecto, Visual Studio no lo hará.

En Visual Studio, puede manejar estos dos trabajos haciendo clic derecho en el nombre de la solución en el Explorador de la solución y seleccionando **Administrar paquetes NuGet para la solución**. Los **Administrar paquetes de soluciones** de diálogo le permite descargar y restaurar los paquetes NuGet y actualizarlos.

En Xamarin de estudio, el proceso es algo más automático, pero también se puede utilizar el **Los paquetes de actualización NuGet y Restaurar Paquetes NuGet** opciones en el **Proyecto** menú.

Algunos de los proyectos contienen referencias a las bibliotecas de la **bibliotecas** carpeta del código de ejemplo. Usted querrá cargar esas soluciones biblioteca en Visual Studio o Xamarin estudio separado y restaurar (o actualizar) los paquetes NuGet. A continuación, cargue los proyectos que hacen referencia a estas bibliotecas.

Expresiones de gratitud

Siempre parecía peculiar para mí que los autores de libros de programación a veces se conocen mejor a los programadores que las personas que realmente crearon el producto que es el tema del libro! El verdadero cerebro detrás de Xamarin.Forms son Jason Smith, Eric Maupin, Stephane Delcroix, Seth Rosetter, Rui Marinho, Chris King, EZ Hart, Samantha Houts, Paul DiPietro, y el gerente de producto intermedio Bryan Hunter. ¡Felicitaciones chicos! Hemos podido disfrutar de los frutos de su trabajo!

Durante los meses que estas diversas ediciones del libro estaban en curso, me he beneficiado de información valiosa, correcciones y ediciones de varias personas. Este libro no existiría sin la colaboración de Bryan Costanich en Xamarin y Devon Musgrave en Microsoft Press. Tanto Bryan y Craig Dunn en Xamarin leer algunos de mis borradores de primeros capítulos y lograron convencerme de tomar un enfoque algo diferente al material. Más tarde, Craig me mantuvo en la pista y examinó los capítulos mientras que John Meade hizo la corrección de estilo. Por primera Edición de Vista previa, Stephane Delcroix en Xamarin y Andy Wigley con Microsoft ofrecen esenciales técnica lee y persistentemente me empujaron a hacer el libro mejor. Rui Marinho era a menudo dispuestos a explorar cuestiones técnicas que tenía. Lector de Albert Mata encontró un número de errores tipográficos.

Casi nada de lo que hago en estos días serían posibles sin la compañía diaria y el apoyo de mi esposa, Deirdre Sinnott.

libros electrónicos gratuitos de Microsoft Press

A partir de descripciones técnicas a información en profundidad, los libros electrónicos gratuitos de Microsoft Press cubren una amplia gama de temas. Estos libros electrónicos están disponibles en formato PDF, EPUB, y Mobi para Kindle, formatos y listas para su descarga en <http://aka.ms/mspressfree>.

Queremos escuchar de ti

En Microsoft Press, su satisfacción es nuestra máxima prioridad y su regeneración nuestro activo más valioso. Por favor, díganos lo que piensa de este libro en <http://aka.ms/tellpress>. Su regeneración va directamente a los editores de Microsoft Press. (Se requiere ninguna información personal.) Gracias de antemano por su ayuda!

Capítulo 1

¿Cómo encajan en Xamarin.Forms?

Hay mucha alegría en la programación. Hay alegría en el análisis de un problema, y lo descomponen en pedazos, se formula una solución, trazar una estrategia, se acerca desde diferentes direcciones, y la elaboración del código. Hay mucho alegría de ver el programa ejecute por primera vez, y luego más alegría en el buceo con entusiasmo de nuevo en el código para que sea mejor y más rápido.

También hay a menudo la alegría en la caza de insectos, para asegurar que el programa se ejecuta sin problemas y de manera previsible. Pocas ocasiones son tan alegre como fin la identificación de un error particularmente recalcitrante y definitivamente erradicarla.

Incluso hay alegría en darse cuenta de que el enfoque original que tomó no es lo mejor. Muchos desarrolladores descubren que han aprendido mucho al escribir un programa, incluyendo que hay una mejor manera de estructurar el código. A veces, un parcial o incluso una reescritura total puede resultar en una mejor aplicación, o simplemente uno que es estructuralmente más coherente y fácil de mantener. El proceso es como estar en hombros de uno, y hay mucha alegría en la consecución de ese punto de vista y el conocimiento.

Sin embargo, no todos los aspectos de la programación son tan alegre. Uno de los trabajos más desagradables de programación se está llevando a un programa de trabajo y volver a escribir en un lenguaje de programación completamente diferente o portarlo a otro sistema operativo con una interfaz de programación de aplicaciones totalmente diferente (API).

Un trabajo como ese puede ser un verdadero rutina. Sin embargo, una reescritura de este tipo puede muy bien ser necesario: una aplicación que ha sido tan popular en el iPhone podría ser aún más popular en los dispositivos Android, y sólo hay una forma de averiguarlo.

Pero aquí está el problema: Como vas a través del código fuente original y moverlo a la nueva plataforma, Cómo se mantiene la misma estructura del programa para que existan las dos versiones en paralelo? ¿O se trata de hacer mejoras y mejoras?

La tentación, por supuesto, es repensar por completo la aplicación y hacer la nueva versión mejor. Pero las nuevas versiones de los dos se separan, más dura será la de mantener en el futuro.

Por esta razón, una sensación de temor impregna la bifurcación de una aplicación en dos. Con cada línea de código que se escribe, se da cuenta de que todos los futuros trabajos de mantenimiento, todas las futuras revisiones y mejoras, se han convertido en dos puestos de trabajo en lugar de uno.

Este no es un problema nuevo. Durante más de medio siglo, los desarrolladores han anhelado la capacidad de escribir un único programa que se ejecuta en múltiples máquinas. Esta es una de las razones por las que los lenguajes de alto nivel se inventaron en primer lugar, y esto es por qué el concepto de "desarrollo multiplataforma" sigue ejerciendo un poderoso atractivo como para los programadores.

desarrollo de aplicaciones móviles multiplataforma

La industria de la computación personal ha experimentado un cambio masivo en los últimos años. todavía existen ordenadores de sobremesa, por supuesto, y que siguen siendo vitales para tareas que requieren teclados y pantallas de gran tamaño: programación, la escritura, se extendió por formación de lámina, seguimiento de datos. Pero gran parte de la computación personal se produce ahora en dispositivos más pequeños, en particular para obtener información rápida, el consumo de medios de comunicación y las redes sociales. Las tabletas y los teléfonos inteligentes tienen un paradigma de la interacción del usuario fundamentalmente diferente basado principalmente en el tacto, con un teclado que aparece sólo cuando sea necesario.

El panorama móvil

Aunque el mercado móvil tiene el potencial para un cambio rápido, actualmente dos principales plataformas de teléfonos y tabletas dominan:

- La familia de Apple de iPhones y iPads, los cuales ejecutan el sistema operativo iOS.
- El sistema operativo Android, desarrollado por Google basado en el kernel de Linux, que funciona en una gran variedad de teléfonos y tabletas.

Cómo el mundo se divide entre estos dos gigantes depende de la forma en que se miden: hay más dispositivos Android actualmente en uso, pero los usuarios de iPhone y iPad son más dedicado y pasan más tiempo con sus dispositivos.

También existe una tercera plataforma de desarrollo móvil, que no es tan popular como iOS y Android, sino que implica una empresa con una larga historia en la industria del ordenador personal:

- Teléfono de Microsoft Windows y Windows 10 móvil.

En los últimos años, estas plataformas se han convertido en una alternativa más atractiva ya que Microsoft ha sido la fusión de las APIs de sus plataformas móviles, tabletas, y de escritorio. Tanto el teléfono de Windows 8.1 y Windows 8.1 se basan en una única llamada API de Windows en tiempo de ejecución (o WinRT), que se basa en Microsoft .NET. Esta única API significa que las aplicaciones específicas para las máquinas de sobremesa, portátiles, tabletas y móviles pueden compartir mucho de su código.

Aún más convincente es la plataforma universal de Windows (UWP), una versión del tiempo de ejecución de Windows que forma la base para Windows 10 y Windows 10 móvil. Una aplicación uwp solo puede dirigirse a cada factor de forma del escritorio al teléfono.

Para los desarrolladores de software, la estrategia óptima es apuntar a algo más que una de estas plataformas. Pero eso no es fácil. Hay cuatro grandes obstáculos:

Problema 1: diferentes paradigmas de interfaz de usuario

Todos los tres plataformas incorporan formas similares de la presentación de la interfaz gráfica de usuario (GUI) y la interacción con el dispositivo a través multitouch, pero hay muchas diferencias de detalle. Cada plataforma tiene

diferentes maneras de navegar por las aplicaciones y páginas, diferentes convenciones para la presentación de los datos, diferentes maneras de invocar y menús de la pantalla, e incluso diferentes enfoques para tocar.

Los usuarios se acostumbran a interactuar con aplicaciones en una plataforma particular y esperan aprovechar ese conocimiento con aplicaciones futuras. Cada plataforma adquiere su propia cultura asociada, y estas convenciones culturales influyen entonces desarrolladores.

Problema 2: Los diferentes entornos de desarrollo

Los programadores de hoy están acostumbrados a trabajar en un entorno de desarrollo integrado sofisticada (IDE). existen tales entornos de desarrollo para las tres plataformas, pero por supuesto que son diferentes:

- Para el desarrollo de iOS, Xcode en el Mac.
- Para el desarrollo de Android, Android Studio en una variedad de plataformas.
- Para el desarrollo de Windows, Visual Studio en el PC.

Problema 3: Diferentes interfaces de programación

Los tres de estas plataformas se basan en diferentes sistemas operativos con diferentes APIs. En muchos casos, las tres plataformas implementan todos los tipos similares de objetos de interfaz de usuario, pero con diferentes nombres.

Por ejemplo, las tres plataformas tienen algo que permite al usuario alternar un valor booleano:

- En el iPhone o el iPad, que es una "vista" llamada UISwitch.
- En los dispositivos Android, se trata de un "widget de" llamada Cambiar.
- En la API de Windows en tiempo de ejecución, es un "control" llamada Interruptor de palanca.

Por supuesto, las diferencias van mucho más allá de los nombres en las interfaces de programación propios.

Problema 4: Diferentes lenguajes de programación

Los desarrolladores tienen cierta flexibilidad en la elección de un lenguaje de programación para cada una de estas tres plataformas, pero, en general, cada plataforma está estrechamente asociada con un lenguaje de programación en particular:

- Objective-C para el iPhone y el iPad
- Java para los dispositivos Android
- C # para Windows

Objective-C, Java y C # son primos de clase, ya que son todos descendientes orientadas a objetos de C, pero se han convertido en lugar primos lejanos.

Por estas razones, una empresa que quiere ver en varias plataformas muy bien podría emplear a tres equipos diferentes, cada programador del equipo cualificados y especializados en un lenguaje y API particular.

Este problema del idioma es particularmente desagradable, pero es el problema que es la más tentadora para resolver: Si usted podría utilizar el mismo lenguaje de programación para estas tres plataformas, al menos podrías compartir algo de código entre las plataformas. Este código compartido probablemente no estaría involucrado con la interfaz de usuario, ya que cada plataforma tiene diferentes APIs, pero bien podría ser el código de aplicación que no toca la interfaz de usuario en absoluto.

Un lenguaje único para estas tres plataformas sin duda sería conveniente. Pero, ¿qué idioma se trata?

El C # y solución .NET

Un cuarto lleno de programadores vendría con una variedad de respuestas a la pregunta que plantea solo, pero es un buen argumento puede ser hecho en favor de C#. Presentado por Microsoft en el año 2000, C# es un nuevo lenguaje de programación, al menos en comparación con Objective-C y Java. Al principio, C# parecía ser un establecimiento inflexible, lenguaje más sencillo, imperativo orientado a objetos, sin duda influenciado por C++ (y Java también), pero con una sintaxis mucho más limpia que C++ y ninguno de los bagajes históricos. Además, la primera versión de C# tenía apoyo a nivel de idioma para propiedades y eventos, que resultan ser tipos de miembros que son particularmente adecuados para las interfaces gráficas de programación del usuario.

Sin embargo, C# se ha dejado de crecer y obtener un mejor paso de los años. El apoyo de los genéricos, las funciones lambda, LINQ, y las operaciones asíncronas ha transformado con éxito de C# de forma que ahora se clasifica apropiadamente como un lenguaje de programación multiparadigma. C# código puede ser tradicionalmente imperativo, o el código puede ser aromatizado con los paradigmas de programación declarativos o funcionales.

Desde su creación, C# ha estado estrechamente asociado con el Microsoft .NET Framework. En el nivel más bajo, .NET proporciona una **infraestructura para los C# tipos de datos básicos (int, double, string, Etcétera)**. Pero la extensa biblioteca de clases de .NET Framework proporciona soporte para muchas tareas comunes que se encuentran en muchos tipos diferentes de programación. Éstas incluyen:

- Mates
- depuración
- Reflexión
- Colecciones
- La globalización
- Puedo presentar E / S
- Redes

- Seguridad
- enhebrar
- servicios web
- Manejo de datos
- XML y JSON lectura y la escritura

Aquí hay otra gran razón para C # y .NET debe ser considerada como una solución multi-plataforma convincente:

No es sólo hipotético. Es una realidad.

Poco después del anuncio de Microsoft de .NET camino de regreso en junio de 2000, la empresa Ximian (fundada por Miguel de Icaza y Nat Friedman) inició un proyecto de código abierto llamado Mono para crear una implementación alternativa del compilador de C # y .NET Framework que podría ejecutar en Linux.

Una década más tarde, en 2011, los fundadores de la leche de limpieza (que había sido adquirida por Novell) fundada Xamarin, que sigue contribuyendo a la versión de código abierto del Mono, pero que también se ha adaptado Mono para formar la base de las soluciones móviles multiplataforma.

El año 2014 vio algunos desarrollos en C # y .NET que son un buen augurio para su futuro. Una versión de código abierto del compilador de C #, denominada plataforma .NET Compiler (antes conocido por su nombre en clave "Roslyn") ha sido publicado. Y la Fundación .NET se anunció para servir como administrador de tecnologías de código abierto .NET, en el que Xamarin juega un papel importante.

En marzo de 2016, Microsoft adquirió Xamarin con el objetivo de llevar el desarrollo móvil de plataforma cruzada a la comunidad más amplia de desarrolladores de Microsoft. Xamarin.Forms ahora está libremente disponible para todos los usuarios de Visual Studio.

Un lenguaje único para todas las plataformas

Durante los tres primeros años de su existencia, Xamarin centra principalmente en tecnologías de compilación y tres conjuntos básicos de las bibliotecas .NET:

- Xamarin.Mac, que ha evolucionado a partir del proyecto MonoMac.
- Xamarin.iOS, que evolucionó a partir de MonoTouch.
- Xamarin.Android, que evolucionó a partir de Mono para Android o (más informal) MonoDroid.

En conjunto, estas bibliotecas se conocen como la plataforma Xamarin. Las bibliotecas consisten en versiones de .NET de los nativos de Mac, iOS y Android API. Los programadores que utilizan estas bibliotecas pueden escribir aplicaciones en C # para dirigirse a las API nativas de estas tres plataformas, sino también (como un bono) con acceso a la biblioteca de clases de .NET Framework.

Los desarrolladores pueden utilizar Visual Studio para crear aplicaciones Xamarin, apuntando a iOS y Android, así como todas las diferentes plataformas de Windows. Sin embargo, el iPhone y el desarrollo iPad también requiere un Mac conectado a la PC a través de una red local. Este Mac debe tener instalado Xcode, así como Xamarin Studio, un entorno de desarrollo integrado basado en OS X que le permite desarrollar iPhone, iPad, Mac OS

X, Android y aplicaciones en el Mac. Xamarin Studio no le permiten orientar las plataformas de Windows.

código de compartir

La ventaja de dirigirse múltiples plataformas con un solo lenguaje de programación proviene de la capacidad de compartir código entre las aplicaciones.

Antes de código puede ser compartida, una aplicación debe ser estructurado para tal fin. En particular, desde el uso generalizado de interfaces gráficas de usuario, los programadores han entendido la importancia de separar el código de aplicación en capas funcionales. Tal vez la división más útil es entre el código de interfaz de usuario y los modelos de datos subyacentes y algoritmos. El popular MVC (Modelo-Vista-Controlador) arquitectura de la aplicación formaliza esta separación código en un modelo (los datos subyacentes), la vista (la representación visual de los datos), y el controlador (que se ocupa de la entrada del usuario).

MVC se originó en la década de 1980. Más recientemente, el (Model-View-ViewModel), la arquitectura MVVM ha modernizado de manera efectiva MVC basado en GUI modernas. MVVM separa el código en el modelo (los datos subyacentes), la vista (la interfaz de usuario, incluyendo efectos visuales y de entrada), y el modelo de vista (que gestiona paso de datos entre el modelo y la vista).

Cuando un programador desarrolla una aplicación que se dirige a múltiples plataformas móviles, la arquitectura MVVM ayuda a guiar al desarrollador en la separación de código en el específico de la plataforma Ver el código que requiere la interacción con las APIs de la plataforma y el modelo independiente de la plataforma y el modelo de vista.

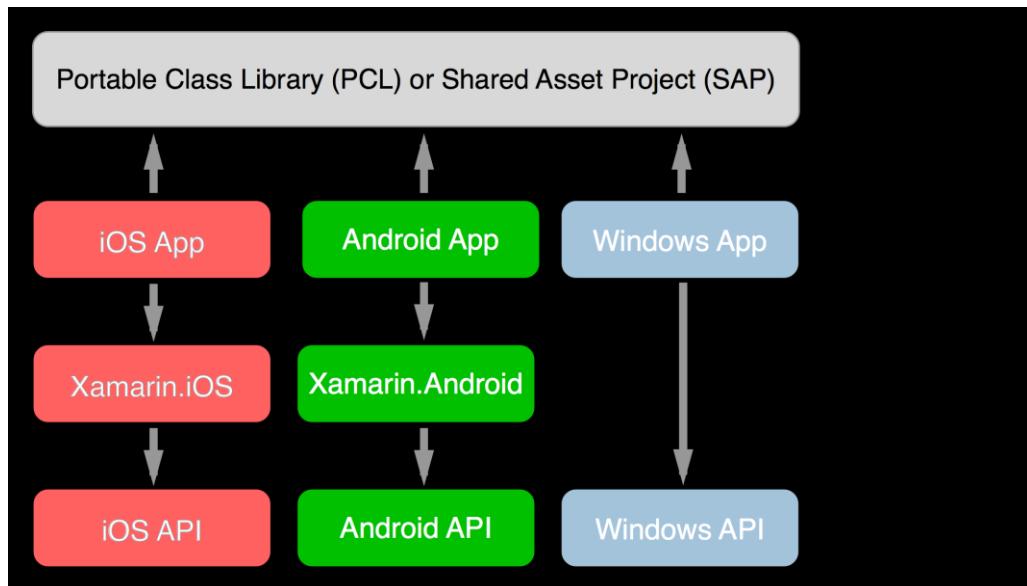
A menudo, este código independiente de la plataforma necesita acceder a archivos o la red o colecciones de uso o roscado. Normalmente estos puestos de trabajo se consideran parte de una API del sistema operativo, sino que también son puestos de trabajo que pueden hacer uso de la biblioteca de clases de .NET Framework, .NET y si está disponible en cada plataforma, a continuación, este código es efectivamente independiente de la plataforma.

La parte de la aplicación que es independiente de la plataforma se puede aislar y en el contexto de Visual Studio o Xamarin Studio-puesto en un proyecto separado. Esto puede ser o bien un proyecto activo compartido (SAP) -que consiste simplemente en archivos de código y otros activos accesible desde otros proyectos o una biblioteca portátil de clase (PCL), que encierra todo el código común en una biblioteca de vínculos dinámicos (DLL) que a continuación, puede hacer referencia a otros proyectos.

Sea cual sea el método que utilice, este código común tiene acceso a la biblioteca de clases de .NET Framework, por lo que puede realizar archivo de E / S, mango de la globalización, los servicios web de acceso, se descomponen XML, y así sucesivamente.

Esto significa que se puede crear una única solución de Visual Studio que contiene cuatro proyectos de C# para apuntar a los tres principales plataformas móviles (todas con acceso a un PCL común o SAP), o puede utilizar Xamarin Studio a los dispositivos de iPhone y Android.

El siguiente diagrama ilustra las interrelaciones entre los proyectos de Visual Studio o Xamarin Studio, las bibliotecas Xamarin, y las APIs de la plataforma. La tercera columna se refiere a cualquier plataforma de Windows basado en .NET independientemente del dispositivo:



Las cajas de la segunda fila son las aplicaciones reales específicos de la plataforma. Estas aplicaciones hacen llamadas en el proyecto común y también (con el iPhone y Android) las bibliotecas Xamarin que implementan las API nativas de la plataforma.

Sin embargo, el diagrama no es bastante completa: no muestra el SAP o PCL realización de llamadas a la biblioteca de clases de .NET Framework. Exactamente qué versión de .NET esto es depende del código común: Un PCL tiene acceso a su propia versión de .NET, mientras que un SAP utiliza la versión de .NET incorporado en cada plataforma en particular.

En este diagrama, las bibliotecas y Xamarin.iOS/Xamarin.Android parecen ser sustanciales, y mientras que son ciertamente importantes, la mayoría son simplemente enlaces de lenguaje y no aumentan significativamente ninguna sobrecarga de llamadas a la API.

Cuando la aplicación de iOS se construye, el Xamarin compilador de C# genera C# lenguaje intermedio (IL), como de costumbre, pero luego hace uso del compilador de Apple en el Mac para generar código nativo de máquina de iOS al igual que el compilador de Objective-C. Las llamadas desde la aplicación a las API de iOS son los mismos que a pesar de la aplicación fueron escritos en Objective-C.

Para la aplicación para Android, el Xamarin compilador de C # genera IL, que se ejecuta en una versión de Mono en el dispositivo junto con el motor de Java, pero las llamadas a la API de la aplicación son más o menos lo mismo que si la aplicación fueron escritos en Java.

Para aplicaciones móviles que tienen necesidades muy específicas de la plataforma, sino también un trozo de código potencialmente compatibles independiente de la plataforma, y Xamarin.iOS Xamarin.Android proporcionan excelentes soluciones. Usted tiene acceso a toda la API de la plataforma, con todo el poder (y la responsabilidad) que ello implica.

Sin embargo, para aplicaciones que podrían no necesitar bastante tanto la especificidad de la plataforma, hay una alternativa que va a simplificar su vida aún más.

La introducción de Xamarin.Forms

El 28 de mayo de 2014, introdujo Xamarin Xamarin.Forms, lo que le permite escribir código de interfaz de usuario que puede ser compilado para los dispositivos iOS, Android y Windows.

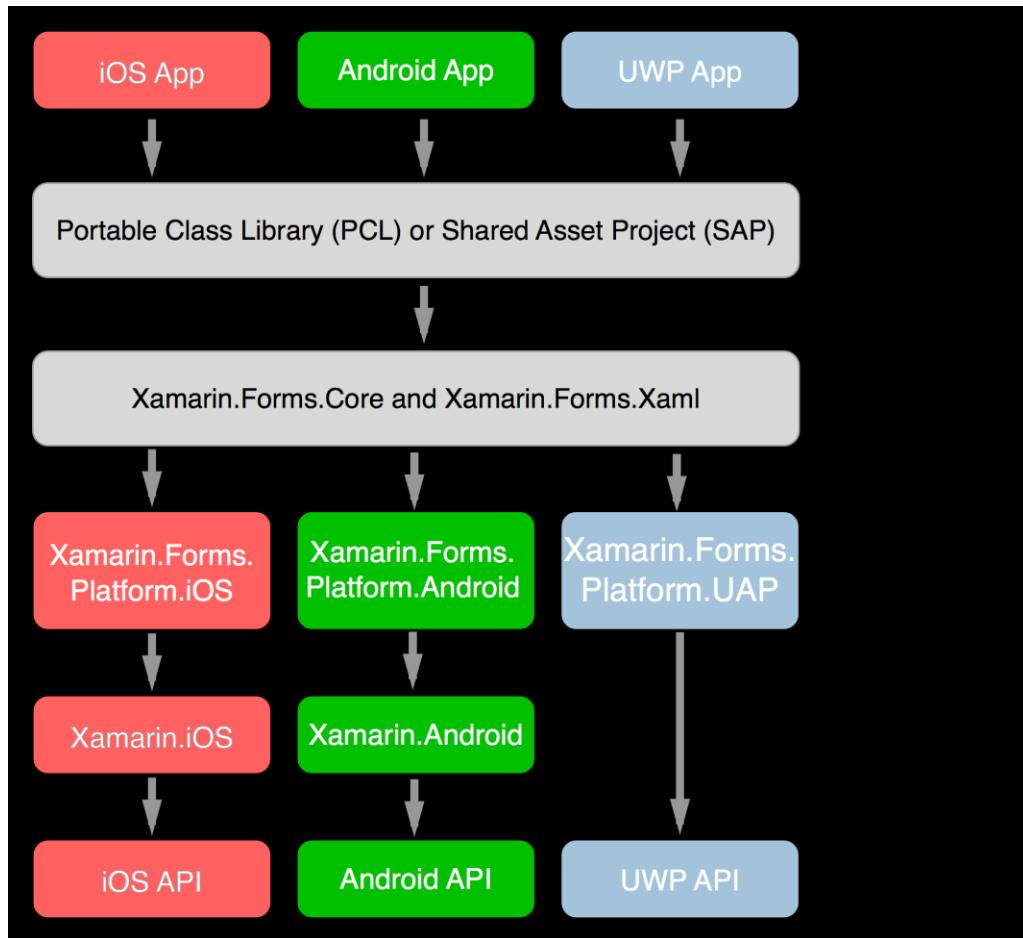
La opción Xamarin.Forms

Xamarin.Forms soporta cinco plataformas de aplicaciones distintas:

- IOS para los programas que se ejecutan en el iPhone, iPad y iPod Touch.
- Android para los programas que se ejecutan en los teléfonos Android y tabletas.
- La plataforma Windows universal (UWP) para las aplicaciones que se ejecuta en Windows 10 o Windows 10 móvil.
- La API de Windows en tiempo de ejecución de Windows de 8.1.
- La API de Windows en tiempo de ejecución de Windows Phone 8.1.

En este libro, "Windows" o "Windows Phone" generalmente se usa como un término genérico para describir los tres de las plataformas de Microsoft.

En el caso general, una aplicación Xamarin.Forms en Visual Studio consta de cinco proyectos separados para cada uno de estos cinco plataformas, con un sexto proyecto que contiene código común. Pero los cinco proyectos de plataforma en una aplicación Xamarin.Forms suelen ser muy pequeños a menudo consiste en stubs con un poco de código de inicio repetitivo. El PCL o SAP contiene la mayor parte de la aplicación, incluyendo el código UserInterface. El siguiente diagrama muestra sólo el IOS, Android y la plataforma de Windows universal. Las otras dos plataformas de Windows son similares a uwp:



Los **Xamarin.Forms.Core** y **Xamarin.Forms.Xaml** bibliotecas implementan la API Xamarin.Forms. Dependiendo de la plataforma, **Xamarin.Forms.Core** a continuación, hace uso de una de las **Xamarin.Forms.Platform** bibliotecas. Estas bibliotecas son en su mayoría una colección de clases llamada *extracción de grasas* que transforman las Xamarin.Forms objetos de interfaz de usuario en la interfaz de usuario específico de la plataforma.

El resto del diagrama es el mismo que el mostrado anteriormente.

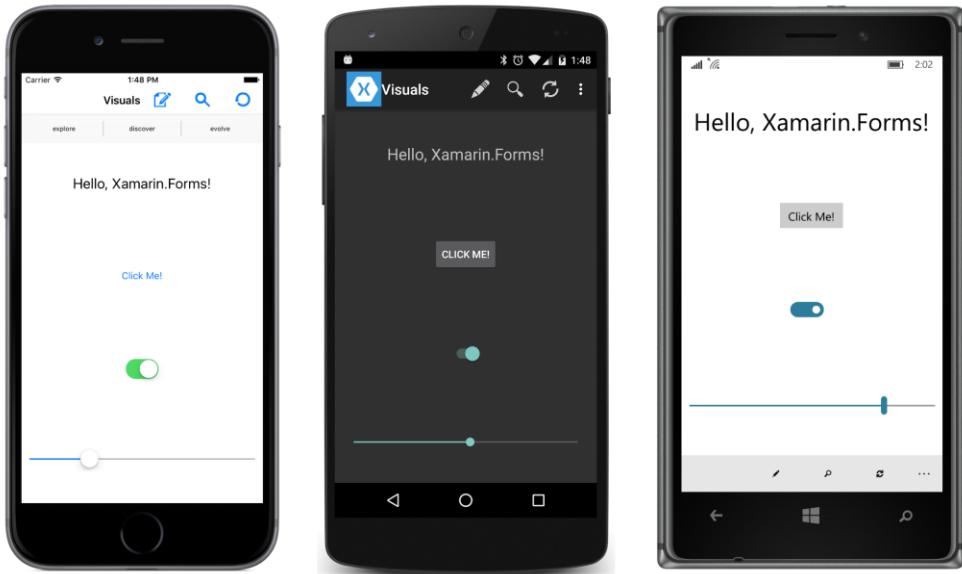
Por ejemplo, supongamos que necesita el objeto de interfaz de usuario se señaló anteriormente que permite al usuario cambiar un valor booleano. Al programar para Xamarin.Forms, esto se llama una Cambiar, y una clase llamada Cambiar se implementa en el **Xamarin.Forms.Core** biblioteca. En los procesadores individuales para las tres plataformas, esta Cambiar se asigna a una UISwitch en el iPhone, una Cambiar en Android, y una

Interruptor de palanca en Windows Phone.

Xamarin.Forms.Core también contiene una clase llamada deslizador para la visualización de una barra horizontal que el usuario manipula para elegir un valor numérico. En los procesadores de las bibliotecas específicas de la plataforma, este se asigna a una UISlider en el iPhone, una Barra de búsqueda en Android, y una deslizador en Windows Phone.

Esto significa que cuando se escribe un programa que tiene un Xamarin.Forms Cambiar o una deslizador, lo que realmente se muestra es el objeto correspondiente implementado en cada plataforma.

He aquí un pequeño programa que contiene una Xamarin.Forms Etiqueta leer "Hola, Xamarin.Forms!", una Botón diciendo "Click Me!", una Cambiar, y una Deslizador. El programa se ejecuta en (de izquierda a derecha) el iPhone, un teléfono Android, y un dispositivo móvil de Windows 10:



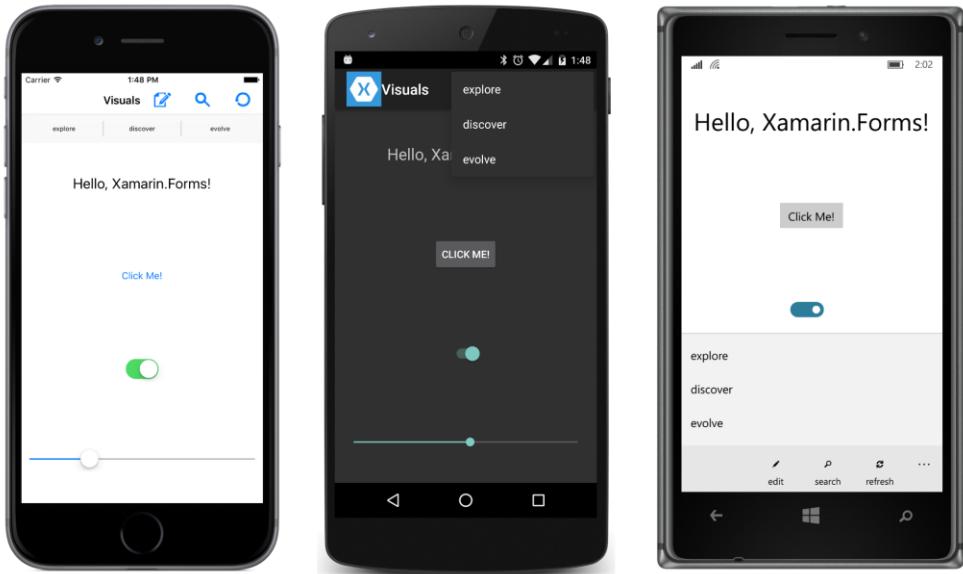
La captura de pantalla iPhone es de un simulador de iPhone 6 con iOS 9.2. El teléfono Android es un LG Nexus 5 con Android versión 6. El Windows 10 es un dispositivo móvil Nokia Lumia 935 ejecutando una vista previa técnica de Windows 10.

Te vas a encontrar imágenes de triples como éste lo largo de este libro. Siempre están en el mismo orden, iPhone, Android y Windows Mobile-10 y que están siempre corriendo el mismo programa.

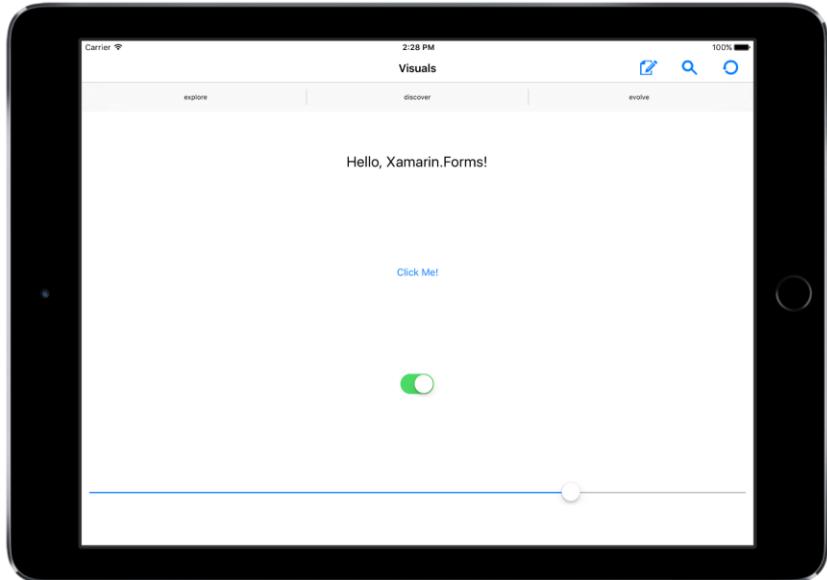
Como se puede ver, el Botón, interruptor, y deslizador todos tienen diferentes aspectos en los tres teléfonos porque todos están representados con el objeto específico para cada plataforma.

Lo que es aún más interesante es la inclusión en este programa de seis ToolbarItem objetos, tres identificados como elementos primarios con iconos, y tres como elementos secundarios sin iconos. En el iPhone éstos son prestados con UIBarButtonItem objetos como los tres iconos y tres botones en la parte superior de la página. En el Android, los tres primeros se representan como elementos en una Barra de acciones. También en la parte superior de la página. En Windows 10 móvil, que están realizados como elementos de la Barra de comando en la parte inferior de la página.

el androide Barra de acciones tiene una elipse vertical y la plataforma Windows universal Barra de comando tiene una elipsis horizontal. Un toque a este elipsis hace que los elementos secundarios que se mostrarán de una manera apropiada a estas dos plataformas:

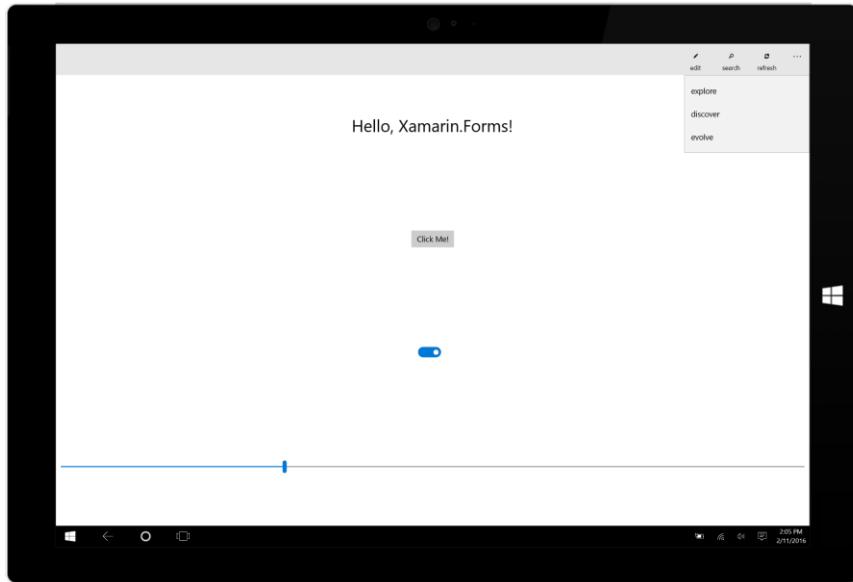


Xamarin.Forms fue concebido originalmente como una API independiente de la plataforma para dispositivos móviles. Sin embargo, Xamarin.Forms no se limita a los teléfonos. Aquí está el mismo programa que se ejecuta en un simulador de iPad Air 2:



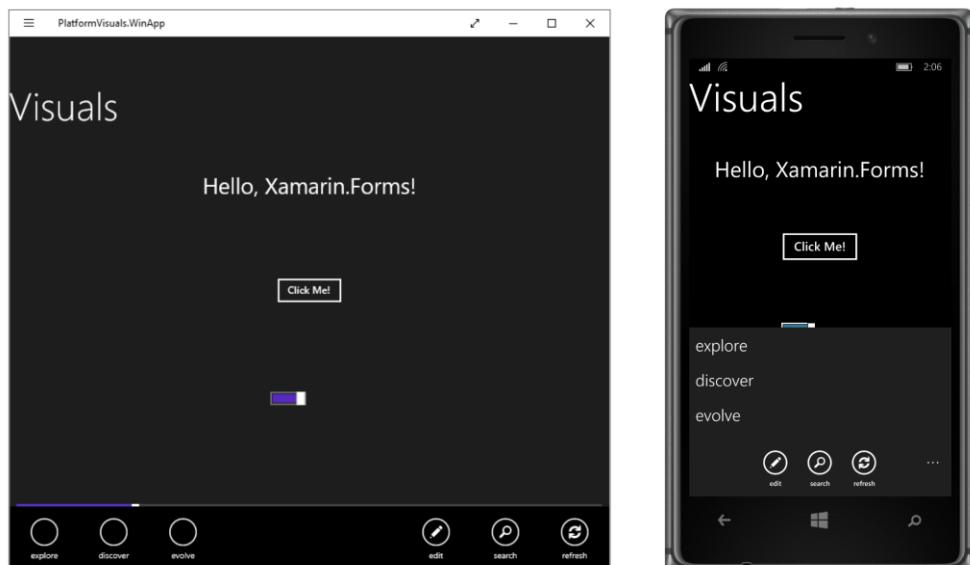
La mayoría de los programas de este libro son bastante simples, y por lo tanto diseñado para lucir lo mejor posible en una pantalla de teléfono en el modo vertical. Pero también se ejecutará en modo horizontal y en las tabletas.

Aquí está el proyecto uwp en una superficie Pro Microsoft Windows 10 3 corriendo:



Observe la barra de herramientas en la parte superior de la pantalla. Los puntos suspensivos ya ha sido presionado para revelar los tres elementos secundarios.

Las otras dos plataformas soportadas por Xamarin.Forms son Windows 8.1 y Windows Phone 8.1. Aquí está el programa de Windows 8.1 que se ejecuta en una ventana en el escritorio de Windows 10 y Windows 8.1 programa que se ejecuta en el dispositivo móvil de Windows 10:



La pantalla de Windows 8.1 se ha hecho clic-izquierda con el ratón para revelar los elementos de la barra en la parte inferior. En esta pantalla, los elementos secundarios están a la izquierda, pero el programa negligentemente olvidaron de asignarlos iconos. En la pantalla de Windows Phone 8.1, los puntos suspensivos en la parte inferior se ha pulsado.

Las diversas implementaciones de la barra de herramientas revela que, en un sentido, Xamarin.Forms es una API que virtualiza no sólo los elementos de interfaz de usuario en cada plataforma, sino también los paradigmas de interfaz de usuario.

apoyo XAML

Xamarin.Forms también es compatible con XAML (pronunciado "zammel" para rimar con "camello"), el basado en XML Extensible Application Markup Language desarrollado en Microsoft como un lenguaje de marcado de propósito general para instanciar e inicializar objetos. XAML no se limita a la definición de los diseños iniciales de interfaces de usuario, pero históricamente que es como se ha utilizado al máximo, y eso es lo que se usa en el Xamarin-.Formularios.

Aquí está el archivo XAML para el programa que tiene como capturas de pantalla que acaba de ver:

```
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns:x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " PlatformVisuals.PlatformVisualsPage "
    Título = " visuales " >

    < StackLayout Relleno = " 10,0 " >
        < Etiqueta Texto = " Hola, Xamarin.Forms! "
            Tamaño de fuente = " Grande "
            VerticalOptions = " CenterAndExpand "
            HorizontalOptions = " Centrar " >

        < Botón Texto = " ¡Haz click en mí! "
            VerticalOptions = " CenterAndExpand "
            HorizontalOptions = " Centrar " />

        < Cambiar VerticalOptions = " CenterAndExpand "
            HorizontalOptions = " Centrar " />

    < deslizador VerticalOptions = " CenterAndExpand " />
</ StackLayout >

< ContentPage.ToolbarItems >
    < ToolbarItem Texto = " editar " Orden = " Primario " >
        < ToolbarItem.Icon >
            < OnPlatform x: TypeArguments = " FileImageSource "
                iOS = " edit.png "
                Androide = " ic_action_edit.png "
                WinPhone = " imágenes / edit.png " />
        </ ToolbarItem.Icon >
    </ ToolbarItem >

    < ToolbarItem Texto = " buscar " Orden = " Primario " >
        < ToolbarItem.Icon >
```

```

<OnPlatform x:TypeArguments = "FileImageSource"
    iOS = "Search.png"
    Android = "ic_action_search.png"
    WinPhone = "Imágenes / feature.search.png" />
</ToolbarItem.Icon>
</ToolbarItem>

<ToolbarItem Texto = "refrescar" Orden = "Primario">
<ToolbarItem.Icon>
<OnPlatform x:TypeArguments = "FileImageSource"
    iOS = "reload.png"
    Android = "ic_action_refresh.png"
    WinPhone = "Imágenes / refresh.png" />
</ToolbarItem.Icon>
</ToolbarItem>

<ToolbarItem Texto = "explorar" Orden = "Secundario" />
<ToolbarItem Texto = "descubrir" Orden = "Secundario" />
<ToolbarItem Texto = "evolucionar" Orden = "Secundario" />
</ContentPage.ToolbarItems>
</Pagina de contenido>

```

A menos que tenga experiencia con XAML, algunos detalles de sintaxis puede ser un poco oscuro. (No se preocupe,. Aprenderá todo sobre ellos más adelante en este libro), pero aún así, se puede ver la Etiqueta, botón, interruptor, y deslizador las etiquetas. En un programa real, el Botón, interruptor, y deslizador probablemente tendría controladores de eventos adjuntos que se llevarían a cabo en un archivo de código C #. Aquí no lo hacen. los VerticalOptions y HorizontalOptions atributos de ayudar en el diseño; que se discuten en el capítulo siguiente.

especificidad plataforma

En la sección de ese archivo XAML que implica la ToolbarItem, también se puede ver una etiqueta llamada OnPlatform. Esta es una de varias técnicas en Xamarin.Forms que permiten la introducción de algunos especificidad plataforma en el código de otra manera independiente de la plataforma o de marcado. Se utiliza aquí porque cada una de las plataformas separadas tiene algo diferentes requisitos de formato y tamaño de imagen asociados con estos iconos.

Existe una instalación similar en código con el Dispositivo clase. Es posible determinar qué plataforma se ejecuta el código en y para elegir los valores u objetos basados en la plataforma. Por ejemplo, puede especificar diferentes tamaños de letra para cada plataforma o ejecutar diferentes bloques de código basados en la plataforma. Es posible que desee permitir al usuario manipular una deslizador para seleccionar un valor en una plataforma, pero elegir un número de un conjunto de valores explícitos en otra plataforma.

En algunas aplicaciones, pueden ser deseables especificidades plataforma más profundas. Por ejemplo, supongamos que su aplicación requiere las coordenadas GPS del teléfono del usuario. Esto no es algo que Xamarin.Forms proporciona, por lo que habrá necesidad de escribir su propio código específico para cada plataforma para obtener esta información.

los DependencyService clase proporciona una manera de hacer esto de una manera estructurada. Se define una interfaz con los métodos que necesita (por ejemplo, IGetCurrentLocation) y luego poner en práctica esa interfaz con una clase en cada uno de los proyectos de plataforma. A continuación, puede llamar a los métodos de la interfaz

a partir de los Xamarin.Forms proyecto casi tan fácilmente como si fuera parte de la API.

Cada uno de los Xamarin.Forms estándar visual objetos, tales como Etiqueta, botón, interruptor, y deslizador -están apoyado por una clase de procesador en los diferentes Xamarin.Forms.Platform bibliotecas. Cada clase de procesador implementa el objeto específico de la plataforma que corresponde al objeto Xamarin.Forms.

Usted puede crear sus propios objetos visuales personalizados con sus propios procesadores personalizados. El objeto personalizado visual va en el proyecto de código común, y los procesadores personalizados go en los proyectos de plataforma individuales. Para que sea un poco más fácil, en general, usted querrá derivar de una clase existente. Dentro de las bibliotecas de la plataforma Xamarin.Forms individuales, todos los procesadores correspondientes son clases públicas, y se puede derivar de ellos también.

Xamarin.Forms le permite ser tan independiente de la plataforma o como plataforma específica como tiene que ser. Xamarin.Forms no reemplaza Xamarin.iOS y Xamarin.Android; más bien, se integra con ellos.

Una panacea multiplataforma?

En su mayor parte, Xamarin.Forms define sus abstracciones con un enfoque en las áreas de la interfaz de usuario móvil que son comunes a los de iOS, Android y tiempo de ejecución de las API de Windows. Estos Xamarin.Forms objetos visuales se asignan a los objetos específicos de la plataforma, pero Xamarin.Forms ha tendido a evitar la aplicación de cualquier cosa que es única para una plataforma en particular.

Por esta razón, a pesar de la enorme ayuda que pueden ofrecer Xamarin.Forms en la creación de aplicaciones platformindependent, no es un reemplazo completo de la programación API nativa. Si su aplicación depende en gran medida de las características de la API nativas tales como determinados tipos de controles o widgets, entonces es posible que desee quedarse con Xamarin.iOS, Xamarin.Android, y la API de Windows Phone nativa.

Es posible que también quiere quedarse con las API nativas para aplicaciones que requieren gráficos vectoriales o la interacción táctil compleja. La versión actual de Xamarin.Forms no está listo para estos escenarios.

Por otro lado, Xamarin.Forms es ideal para la creación de prototipos o hacer una rápida aplicación de prueba de concepto. Y después de que lo haya hecho, que sólo podría encontrar que usted puede seguir utilizando Xamarin.Forms características para construir toda la aplicación. Xamarin.Forms es ideal para aplicaciones de línea de negocio.

Aun cuando empiece la construcción de una aplicación con Xamarin.Forms y luego implementar grandes partes de ella con APIs de la plataforma, por lo que está haciendo en un marco que le permite compartir código y que ofrece formas estructuradas para hacer visuales específicas de la plataforma.

Su entorno de desarrollo

¿Cómo se configura el hardware y el software depende de lo que las plataformas móviles que usted está apuntando y qué entornos informáticos son más cómodo para usted.

Los requisitos para Xamarin.Forms no son diferentes de los requisitos para usar Xamarin.iOS o Xamarin.Android o para la programación para las plataformas de Windows en tiempo de ejecución.

Esto significa que nada en esta sección (y el resto de este capítulo) es específico para Xamarin.Forms. Existe mucha documentación en el sitio web Xamarin sobre la configuración de las máquinas y el software de programación y Xamarin.iOS, Xamarin.Android, y en el sitio web de Microsoft sobre Windows Phone.

Máquinas y entornos de desarrollo

Si desea orientar el iPhone, vas a necesitar un Mac. Apple requiere que un Mac puede utilizar para crear aplicaciones de iPhone y otros IOS. Tendrá que instalar Xcode en esta máquina y, por supuesto, la plataforma Xamarin que incluye las librerías necesarias y Xamarin Studio. A continuación, puede utilizar Xamarin Studio y Xamarin.Forms en el Mac para su desarrollo del iPhone.

Una vez que tenga un Mac con Xcode y la plataforma Xamarin instalado, también puede instalar la plataforma Xamarin en un PC y el programa para el iPhone mediante el uso de Visual Studio. El PC y Mac deben estar conectados a través de una red (tales como Wi-Fi). Visual Studio se comunica con el Mac a través de una interfaz de Secure Shell (SSH), y utiliza el Mac para generar la aplicación y ejecutar el programa en un dispositivo o un simulador.

También se puede hacer la programación en Android Xamarin Studio en el Mac o en Visual Studio en el PC.

Si desea orientar las plataformas Windows, necesitará Visual Studio 2015. Puede orientar todas las plataformas en un solo IDE mediante la ejecución de Visual Studio 2015 en un PC conectado al Mac a través de una red. (Así es como se crearon los programas de ejemplo en este libro.) Otra opción es ejecutar Visual Studio en una máquina virtual en el Mac.

Dispositivos y emuladores

Usted puede probar sus programas en los teléfonos reales conectados a las máquinas a través de un cable USB, o puede probar los programas con los emuladores que aparecen en pantalla.

Hay ventajas y desventajas de cada método. Un teléfono real es esencial para probar la interacción táctil compleja o cuando conseguir una sensación para el inicio o el tiempo de respuesta. Sin embargo, los emuladores permiten ver cómo su aplicación se adapta a una variedad de tamaños y factores de forma.

Los emuladores de iPhone y iPad se ejecutan en el Mac. Sin embargo, debido a las máquinas de escritorio de Mac no tienen pantallas táctiles, tendrá que utilizar el ratón o el trackpad para simular el tacto. Los gestos táctiles en la pantalla táctil Mac no se traducen en el emulador. También puede conectar un iPhone real a la Mac, pero necesitará para la provisión como un dispositivo desarrollador.

Históricamente, los emuladores de Android suministrados por Google han tendido a ser lento y de mal humor, aunque a menudo son extremadamente versátil en la emulación de una amplia gama de dispositivos Android reales. Afortunadamente, Visual Studio ahora tiene su propio emulador de Android que funciona bastante mejor. También es muy fácil de conectar.

un verdadero teléfono Android a un Mac o PC para la prueba. Todo lo que necesitas hacer es habilitar la depuración USB en el dispositivo.

Los emuladores de Windows Phone son capaces de varios diferentes resoluciones de pantalla y también tienden a funcionar bastante bien, aunque consume mucha memoria. Si ejecuta el emulador de Windows Phone en una pantalla táctil, se puede usar el tacto en la pantalla del emulador. **Conexión de un verdadero teléfono de Windows en el PC es bastante fácil, pero requiere que permite que el teléfono en el ajustes sección para el desarrollo.** Si desea desbloquear más de un teléfono, necesitará una cuenta de desarrollador.

Instalación

Antes de escribir aplicaciones para Xamarin.Forms, tendrá que instalar la plataforma Xamarin en su Mac, PC o ambos (si está utilizando esa configuración). Ver los artículos en el sitio web Xamarin en:

https://developer.xamarin.com/guides/cross-platform/getting_started/installation/

Probablemente usted está ansioso por crear su primera aplicación Xamarin.Forms, pero antes de hacerlo, tendrá que intentar la creación de proyectos normales Xamarin para el iPhone y Android y Windows normal, Windows Phone y Windows Mobile 10 proyectos.

Esto es importante: si usted está experimentando un problema usando Xamarin.iOS, Xamarin.Android, o Windows, eso no es un problema con Xamarin.Forms, y que necesita para resolver el problema antes de usar Xamarin.Forms.

La creación de una aplicación para iOS

Si está interesado en utilizar Xamarin.Forms para apuntar el iPhone, primero familiarizarse con los documentos apropiados Getting Started en el sitio web Xamarin:

https://developer.xamarin.com/guides/ios/getting_started/

Esto le dará orientación sobre el uso de la biblioteca Xamarin.iOS para desarrollar una aplicación de iPhone en C#. Todo lo que necesita hacer es llegar al punto donde se puede construir e implementar una aplicación iPhone sencilla ya sea en un iPhone real o el simulador de iPhone.

Si está utilizando Visual Studio, y si todo está instalado correctamente, debería ser capaz de seleccionar **Archivo > Nuevo > Proyecto** en el menú, y en el **Nuevo proyecto** de diálogo, desde la izquierda, seleccione **Visual C# y iOS** y entonces (**universal** que se refiere a la orientación tanto iPhone y iPad), y de la lista de plantillas en el centro, seleccione **App Blank (iOS)**.

Si está utilizando Xamarin de estudio, usted debe ser capaz de seleccionar **Archivo > Nuevo > Solución** en el menú, y en el **Nuevo proyecto** de diálogo, desde la izquierda, seleccione **iOS** y entonces **aplicación**, y la lista de plantillas en el centro, seleccione **Vista única aplicación**.

En cualquier caso, seleccionar una ubicación y el nombre de la solución. Construir y desplegar la aplicación marco creado en el proyecto. Si tiene un problema con esto, no es una cuestión Xamarin.Forms. Es posible que desee revisar los foros Xamarin.iOS para ver si alguien más tiene un problema similar:

<http://forums.xamarin.com/categories/ios/>

La creación de una aplicación para Android

Si está interesado en utilizar Xamarin.Forms a los dispositivos de Android, primero familiarizarse con los documentos Getting Started en el sitio web Xamarin:

https://developer.xamarin.com/guides/android/getting_started/

Si está utilizando Visual Studio, y si todo está instalado correctamente, debería ser capaz de seleccionar **Archivo**

> **Nuevo> Proyecto** en el menú, y en el **Nuevo proyecto** de diálogo, desde la izquierda, seleccione **Visual C# y entonces Androide**, y la lista de plantillas en el centro, seleccione **Aplicación en blanco (Android)**.

Si está utilizando Xamarin de estudio, usted debe ser capaz de seleccionar **Archivo> Nuevo> Solución** en el menú, y en el **Nuevo proyecto** de diálogo, desde la izquierda, seleccione **Androide y aplicación**, y en la lista de plantillas en el centro, seleccione **Aplicación Android**.

Darle un lugar y un nombre; construir y desplegar. Si usted no puede conseguir que este proceso funcione, no es una cuestión Xamarin.Forms, y es posible que desee comprobar los foros Xamarin.Android para un problema similar:

<http://forums.xamarin.com/categories/android/>

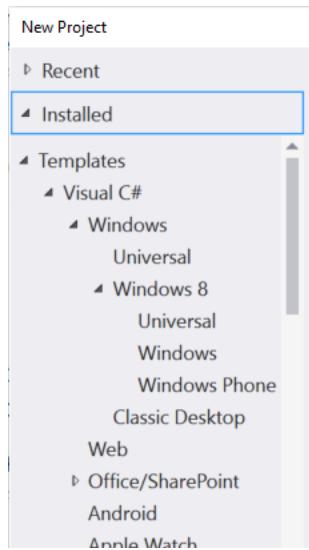
Creación de una aplicación de Windows

Si está interesado en utilizar para apuntar Xamarin.Forms de Windows, Windows Phone o Windows Mobile 10, tendrá que familiarizarse con al menos los rudimentos de uso de Visual Studio para desarrollar aplicaciones de Windows:

<http://dev.windows.com/>

En Visual Studio 2015, si todo está correctamente instalado, usted debería ser capaz de selección **Archivo> Nuevo> Proyecto** en el menú, y en el **Nuevo proyecto** de diálogo, a la izquierda, seleccione **Visual C# y Ventanas**.

Usted verá una jerarquía bajo la **ventanas** rúbrica algo como esto:



El primero **Universal** rúbrica bajo **ventanas** es una aplicación para la creación de la plataforma de Windows universal que puede apuntar ya sea Windows 10 o Windows 10 móvil. Seleccionar eso, y desde la zona central de selección **Aplicación en blanco (Universal Windows)** para crear una aplicación UWP.

Los otros dos tipos de proyectos apoyados por Xamarin.Forms están bajo el encabezado de Windows 8. los **Universal** elemento realidad crea dos proyectos: una aplicación de escritorio de Windows y una aplicación de Windows Phone con algo de código compartido. Para la creación de sólo una aplicación de Windows, seleccione **ventanas** y, a continuación de la sección central **Aplicación en blanco (Windows 8.1)**. Para una aplicación de Windows Phone, seleccione **Teléfono windows y Aplicación en blanco** Esto crea un proyecto que apunta a Windows Phone 8.1.

Estos son los tres tipos de proyectos apoyados por Xamarin.Forms.

Usted debe ser capaz de construir y desplegar la aplicación marco en el escritorio o en un teléfono real o un emulador. Si no es así, buscar el sitio web de Microsoft o foros en línea, tales como el desbordamiento de pila.

¿Listo?

Si usted puede construir Xamarin.iOS, Xamarin.Android, y aplicaciones de Windows (o algún subconjunto de los), entonces usted está listo para crear su primera aplicación Xamarin.Forms. Es hora de decir "hola", Xamarin.Forms a una nueva era en el desarrollo móvil de plataforma cruzada.

Capítulo 2

Anatomía de una aplicación

La interfaz de usuario moderna se construye a partir de los objetos visuales de diversos tipos. Dependiendo del sistema operativo, estos objetos visuales podrían ir por diferentes nombres, controles, elementos, puntos de vista, widgets- pero todos ellos son dedicados a los trabajos de presentación o interacción o ambos.

En Xamarin.Forms, los objetos que aparecen en la pantalla se denominan colectivamente *elementos visuales*. Vienen en tres categorías principales:

- página
- diseño
- ver

Estos no son conceptos abstractos! La interfaz de programación de aplicaciones Xamarin.Forms (API) define las clases nombradas VisualElement, página, diseño, y Ver. Estas clases y sus descendientes forman la columna vertebral de la interfaz Xamarin.Forms usuario. VisualElement es una clase de excepcional importancia en Xamarin.Forms. UN VisualElement objeto es cualquier cosa que ocupa espacio en la pantalla.

Una aplicación Xamarin.Forms consiste en una o más páginas. Una página suele ocupar todos (o al menos una gran superficie) de la pantalla. Algunas aplicaciones consisten en una única página, mientras que otros permiten navegar entre múltiples páginas. En muchos de los primeros capítulos de este libro, verá sólo un tipo de página, llamada Pagina de contenido.

En cada página, los elementos visuales se organizan en una jerarquía de elementos primarios y secundarios. El hijo de una Estaña-tentPage es por lo general un diseño de algún tipo para organizar los elementos visuales. Algunos diseños tienen un solo niño, pero muchos diseños tener varios hijos que la disposición organiza dentro de sí mismo. Estos niños pueden ser otros diseños o puntos de vista. Los diferentes tipos de diseños de organizar los niños en una pila, en una cuadricula de dos dimensiones, o de una manera más libre. En este capítulo, sin embargo, nuestras páginas contendrán un solo hijo.

El término *ver* en Xamarin.Forms denota tipos conocidos de presentación e interactivos objetos: texto, mapas de bits, botones, campos de entrada de texto, deslizadores, interruptores, barras, fecha y hora recolectores, y otros de su propia invención progresar. Estos son a menudo llamados controles o widgets en otros entornos de programación. Este libro se refiere a ellos como vistas o elementos. En este capítulo, se encontrará con la Etiqueta ver para mostrar texto.

Di hola

Utilizando Microsoft Visual Studio o Xamarin de estudio, vamos a crear una nueva aplicación Xamarin.Forms mediante el uso de una plantilla estándar. Este proceso crea una solución que contiene hasta seis proyectos: cinco plataformas

proyectos para iOS, Android, la plataforma Windows universal (UWP), Windows 8.1 y Windows Phone 8,1 y un proyecto común para la mayor parte de su código de aplicación.

En Visual Studio, seleccione la opción de menú **Archivo> Nuevo> Proyecto**. A la izquierda de la **Nuevo proyecto** diálogo, seleccione **Visual C# y entonces Cruz-plataforma**. En la parte central del cuadro de diálogo verá varias plantillas de soluciones disponibles, incluyendo tres para Xamarin.Forms:

- En blanco de la aplicación (Xamarin.Forms Portable)
- Blank App (Xamarin.Forms Compartido)
- Biblioteca de clases (Xamarin.Forms)

¿Ahora que? Definitivamente, queremos crear una **Aplicación en blanco** solución, pero ¿qué tipo?

Xamarin estudio presenta un dilema similar, pero de una manera diferente. Para crear una nueva solución Xamarin.Forms en Xamarin Studio, seleccione **Archivo> Nuevo> Solución** en el menú, y en la parte izquierda de la **Nuevo proyecto** de diálogo, en **Multiplataforma seleccionar aplicación, recoger Formas de aplicación**, y pulse el **Siguiente** botón. Hacia la parte inferior de la pantalla siguiente son un par de botones de radio marcado **Código Compartido**. Estos botones le permiten elegir una de las siguientes opciones:

- Utilizar la biblioteca de clases portátil
- El uso de biblioteca compartida

El término "portátil" en este contexto se refiere a una biblioteca de clases portátil (PCL). Todo el código de aplicación común se convierte en una biblioteca de vínculos dinámicos (DLL) que hace referencia a todos los proyectos de plataforma individuales.

El término "compartido" en este contexto significa un proyecto activo compartido (SAP) que contiene archivos sueltos de código (y tal vez otros archivos) que son compartidos entre los proyectos de plataforma, convirtiéndose en parte esencial de cada proyecto de plataforma.

Por ahora, elegir la primera de ellas: **En blanco de la aplicación (Xamarin.Forms Portable)** en Visual Studio o **Utilizar la biblioteca de clases portátil** en Xamarin Studio. Dar un nombre al proyecto, por ejemplo, **Hola** -Y seleccionar una ubicación de disco para que en ese diálogo (en Visual Studio) o en el cuadro de diálogo que aparece después de pulsar el **Siguiente** botón de nuevo en Xamarin Studio.

Si se está utilizando Visual Studio, se crean seis proyectos: un proyecto común (el proyecto PCL) y cinco proyectos de aplicación. Para una solución llamada **Hola**, estos son:

- Un proyecto de biblioteca de clases portátil llamado **Hola** que hace referencia a los cinco proyectos de aplicación;
- Un proyecto de aplicación para Android, llamado **Hello.Droid**;
- Un proyecto de aplicación para iOS, llamada **Hello.iOS**;
- Un proyecto de aplicación para la plataforma de Windows universal de Windows 10 y Windows Mobile 10, de nombre **Hello.UWP**;

- Un proyecto de aplicación para Windows 8.1, llamado **Hello.Windows**; y
- Un proyecto de aplicación para Windows Phone 8.1, llamado **Hello.WinPhone**.

Si está ejecutando Xamarin Estudio en el Mac, no se crean los proyectos de Windows y Windows Phone.

Cuando se crea una nueva solución Xamarin.Forms, las bibliotecas Xamarin.Forms (y varias bibliotecas de apoyo) se descargan automáticamente desde el gestor de paquetes NuGet. Visual Studio y Studio Xamarin almacenar estas bibliotecas en un directorio llamado **paquetes** en el directorio de solución. Sin embargo, la versión particular de la biblioteca Xamarin.Forms que se descarga se especifica dentro de la plantilla de la solución, y una versión más reciente podría estar disponible.

En Visual Studio, en el **Explorador de la solución** en el extremo derecho de la pantalla, haga clic en el nombre de la solución y seleccione **Administrar paquetes NuGet para la solución**. El cuadro de diálogo que aparece contiene elementos seleccionables en la parte superior izquierda que le permiten ver lo que se instalan los paquetes NuGet en la solución y permitirán instalar otros. También puede seleccionar el **Actualizar** elemento para actualizar la biblioteca Xamarin.Forms.

En Xamarin.Studio, puede seleccionar el ícono de la herramienta a la derecha del nombre de la solución en el **Solución** lista y seleccionar **Actualizar paquetes NuGet**.

Antes de continuar, verifique para asegurarse de que las configuraciones del proyecto están bien. En Visual Studio, seleccione el **Construir> Administrador de configuración** opción del menú. En el **Administrador de configuración** de diálogo, verá el proyecto PCL y los cinco proyectos de aplicación. Asegúrate que **Construir** casilla está marcada para todos los proyectos y la **Desplegar** casilla está marcada para todos los proyectos de aplicación (a menos que el cuadro esté en gris). Toma nota de la **Plataforma** columna: Si el **Hola** proyecto está en la lista, debe ser marcado como **Cualquier CPU**. Los

Hello.Droid proyecto también debe ser marcado como **Cualquier CPU**. (Para esos dos tipos de proyectos, **cualquier CPU** es la única opción.) Para el **Hello.iOS** proyecto, elija **iPhone** o **iPhoneSimulator** dependiendo de cómo va a ser probar el programa.

Para el **Hello.UWP** proyecto, la configuración del proyecto debe ser **x86** para el despliegue en el escritorio de Windows o un emulador de la pantalla, y **BRAZO** para desplegar a un teléfono.

Para el **Hello.WinPhone** proyecto, puede seleccionar **x86** Si va a utilizar un emulador de pantalla en, **BRAZO** si usted va a desplegar en un teléfono real, o **cualquier CPU** para el despliegue a cualquiera. Independientemente de su elección, Visual Studio genera el mismo código.

Si no parece ser la compilación o desplegando en Visual Studio un proyecto, vuelva a comprobar los ajustes en el **Administrador de configuración** diálogo. A veces una configuración diferente se activa y podría no incluir el proyecto PCL.

En Xamarin Studio en el Mac, puede cambiar entre la implementación en el simulador de iPhone y el iPhone a través de la **Proyecto> Configuración activa** opción del menú.

En Visual Studio, es probable que desee para mostrar las barras de herramientas de iOS y Android. Estas barras de herramientas le permiten elegir entre los **emuladores y dispositivos** y permiten administrar los **emuladores**. En el menú principal, asegúrese de que la **Ver> Barras de herramientas> IOS** y **Ver> Barras de herramientas> Android** los artículos se comprueban.

Debido a que la solución contiene de dos a seis proyectos, debe designar qué programa se pone en marcha cuando se opta por ejecutar o depurar una aplicación.

En el **Explorador de la solución** de Visual Studio, haga clic en cualquiera de los cinco proyectos de aplicación y selección la **Establecer como proyecto de inicio** elemento del menú. A continuación, puede seleccionar para desplegar ya sea a un emulador o un dispositivo real. Para generar y ejecutar el programa, seleccionar la opción de menú **Depurar> Iniciar depuración**.

En el **Solución Lista** de Xamarin Studio, haga clic en el ícono de la herramienta pequeña que aparece a la derecha de un proyecto seleccionado y seleccione **Establecer como proyecto de inicio** en el menú. A continuación, puede elegir **Ejecutar> Iniciar depuración** en el menú principal.

Si todo va bien, la aplicación marco creado por la plantilla se ejecutará y verá un mensaje corto:



Como se puede ver, estas plataformas tienen diferentes combinaciones de colores. El iOS y Windows 10 pantallas móvil de la visualización de texto oscuro sobre un fondo claro, mientras que el dispositivo Android muestra texto claro sobre un fondo negro. Por defecto, las plataformas de Windows 8.1 y Windows Phone 8.1 son como Android en la visualización de texto claro sobre un fondo negro.

Por defecto, todas las plataformas están habilitados para los cambios de orientación. Gire el teléfono hacia los lados, y verá el texto se ajuste a la nueva central.

La aplicación no sólo se ejecuta en el dispositivo o emulador pero desplegado. Aparece con las otras aplicaciones en el teléfono o en el emulador y se puede ejecutar desde allí. Si no te gusta el ícono de la aplicación o cómo las pantallas nombre de la aplicación, se puede cambiar eso en los proyectos de plataforma individuales.

Dentro de los archivos

Claramente, el programa creado por la plantilla Xamarin.Forms es muy simple, por lo que esta es una excelente oportunidad para examinar los archivos de código generado y averiguar sus interrelaciones y cómo funcionan.

Vamos a empezar con el código que es responsable de elaborar el texto que ve en la pantalla. Este es el Aplicación clase en el Hola proyecto. En un proyecto creado por Visual Studio, el Aplicación clase se define en el archivo App.cs, pero en Xamarin Studio, el archivo es Hello.cs. Si la plantilla de proyecto no ha cambiado demasiado ya que este capítulo fue escrito, es probable que se ve algo como esto:

```
utilizando Sistema;
utilizando System.Collections.Generic;
utilizando System.Linq;
utilizando System.Text;

utilizando Xamarin.Forms;

espacio de nombres Hola
{
    clase pública Aplicación : Solicitud
    {
        público App ()
        {
            // La página raíz de su solicitud
            MainPage = nuevo Página de contenido
            {
                content = nuevo StackLayout
                {
                    VerticalOptions = LayoutOptions.Centrar,
                    Los niños =
                    {
                        nuevo Etiqueta {
                            HorizontalTextAlignment = Alineación del texto.Centrar,
                            text = "Bienvenido a las formas Xamarin!"
                        }
                    }
                };
            }

            protegido override void OnStart ()
            {
                // manipulador cuando se inicia el app
            }

            protegido override void OnSleep ()
            {
                // manipulador cuando su aplicación duerme
            }

            protegido override void En resumen()
```

```
{  
    // manejar cuando sus hojas de vida de aplicaciones  
}  
}  
}
```

Observe que el espacio de nombres es el mismo que el nombre del proyecto. Esta Aplicación la clase se define como pública y se deriva de los **Xamarin.Forms.Solicitud** clase. El constructor tiene realmente sólo una responsabilidad: para establecer el **Página principal** propiedad de la **Solicitud** clase a un objeto de tipo Página.

El código que la plantilla Xamarin.Forms ha generado aquí una muestra enfoque muy simple para definir este constructor: la **Página de contenido** clase se deriva de **Página** y es muy común en las aplicaciones Xamarin.Forms **SinglePage**. (Usted verá un montón de **Página de contenido** a lo largo de este libro.) Ocupa la mayor parte de la pantalla del teléfono con la excepción de la barra de estado en la parte superior de la pantalla de Android, los botones de la parte inferior de la pantalla de Android, y la barra de estado en la parte superior de la pantalla de Windows Phone. (Como usted descubrirá, la barra de estado IOS es en realidad parte de la **Página de contenido** en aplicaciones de una sola página.)

Los **Página de contenido** clase define una propiedad denominada **Contenido** que se establece con el contenido de la página. En general, este contenido es un diseño que a su vez contiene un montón de puntos de vista, y en este caso se configura en una **StackLayout**, que organiza sus hijos en una pila.

Esta **StackLayout** tiene un solo hijo, que es una **Etiqueta**. Los **Etiqueta** clase se deriva de **Ver** y se utiliza en aplicaciones Xamarin.Forms de mostrar hasta un párrafo de texto. Los **VerticalOptions** y **HorizontalTextAlignment** propiedades se discuten con más detalle más adelante en este capítulo.

Para sus propias aplicaciones de una sola página Xamarin.Forms, por lo general, estará definiendo su propia clase que deriva de **Página de contenido**. El constructor de la **Aplicación** clase a continuación, establece una instancia de la clase que se define a su **Página principal** propiedad. Vas a ver cómo funciona esto en breve.

En el **Hola** solución, también verá un archivo **AssemblyInfo.cs** para la creación de la PCL y un archivo **packages.config** que contiene los paquetes NuGet requeridos por el programa. En el **referencias** sección bajo **Hola** en la lista de solución, verá al menos las cuatro bibliotecas de esta PCL requiere:

- .NET (se muestra como subconjunto portátil .NET en Xamarin Studio)
- **Xamarin.Forms.Core**
- **Xamarin.Forms.Xaml**
- **Xamarin.Forms.Platform**

Es este proyecto PCL que recibirá la mayor parte de su atención mientras está escribiendo una aplicación Xamarin.Forms. En algunas circunstancias, el código de este proyecto podría requerir cierto ajuste de las diversas plataformas, y verá en breve cómo hacer eso. También puede incluir código específico de la plataforma en los cinco proyectos de aplicación.

Los cinco proyectos de aplicaciones tienen sus propios activos en forma de iconos y metadatos, y se debe prestar especial atención a estos activos si tiene la intención de llevar la aplicación al mercado. Pero durante el tiempo que usted está aprendiendo cómo desarrollar aplicaciones utilizando Xamarin.Forms, estos activos pueden generalmente ser ignorados. Es probable que desee mantener estos proyectos de colapsaron en la lista de solución, ya que no tiene que molestarte mucho con su contenido.

Pero que realmente debe saber lo que hay en estos proyectos de aplicación, así que vamos a echar un vistazo más de cerca.

En el **referencias** sección de cada proyecto de aplicación, verá referencias al proyecto común (PCL **Hola** en este caso), así como varios ensamblados .NET, los Xamarin.Forms monta los enumerados anteriormente, y los conjuntos Xamarin.Forms adicionales aplicables a cada plataforma:

- **Xamarin.Forms.Platform.Android**
- **Xamarin.Forms.Platform.iOS**
- **Xamarin.Forms.Platform.UAP** (no se muestra explícitamente en el proyecto UWP)
- **Xamarin.Forms.Platform.WinRT**
- **Xamarin.Forms.Platform.WinRT.Tablet**
- **Xamarin.Forms.Platform.WinRT.Phone**

Cada una de estas bibliotecas define una estática **Forms.Init** método en el **Xamarin.Forms** espacio de nombres que inicializa el sistema Xamarin.Forms para esa plataforma en particular. El código de inicio en cada plataforma tiene que hacer una llamada a este método.

También usted ha visto que el proyecto PCL deriva una clase pública denominada **Aplicación** que deriva de **Solicitud**. El código de inicio en cada plataforma también debe crear una instancia de este **Aplicación** clase.

Si está familiarizado con iOS, Android, Windows Phone o desarrollo, es posible que la curiosidad de ver cómo el código de inicio plataforma se encarga de estos puestos de trabajo.

El proyecto IOS

Un proyecto IOS contiene típicamente una clase que deriva de **UIApplicationDelegate**. Sin embargo, la biblioteca **Xamarin.Forms.Platform.iOS** define una clase base alternativa llamada **FormsApplicationDelegate**. En el **Hello.iOS** proyecto, verá este archivo **AppDelegate.cs**, aquí despojados de toda la extraña utilizando directivas y comentarios:

```
utilizando Fundación;
utilizando UIKit;

espacio de nombres Hello.iOS
{
    [ Registro ("AppDelegate" )]
    público clase parcial AppDelegate :
        global :: Xamarin.Forms.Platform.iOS. FormsApplicationDelegate
```

```

    {
        público bool anulación FinishedLaunching ( UIApplication aplicación, NSDictionary opciones)
        {
            global :: Xamarin.Forms. formas .En eso();
            LoadApplication ( nuevo Aplicación ());

            base de retorno .FinishedLaunching (app, opciones);
        }
    }
}

```

Los FinishedLaunching anulación comienza llamando al Forms.Init método definido en la **Xamarin.Forms.Platform.iOS** montaje. A continuación, llama una LoadApplication método (definido por el FormsApplicationDelegate), pasando a ella una nueva instancia de la Aplicación clase definida en el Hola espacio de nombres en el PCL compartido. El objeto de página se establece en el Pagina principal propiedad de este Aplicación objeto se puede usar entonces para crear un objeto de tipo UIViewController, que es el encargado de mostrar el contenido de la página.

El proyecto Android

En la aplicación de Android, el típico Actividad principal clase debe ser derivado de una clase llamada **Xamarin.Forms.FormsApplicationActivity**, se define en la **Xamarin.Forms.Platform.Android** montaje, y el Forms.Init llamada requiere alguna información adicional:

```

utilizando Aplicación Android;
utilizando Android.Content.PM;
utilizando Android.OS;

espacio de nombres Hello.Droid
{
    [Actividad (Label = "Hola" , Icono = "@ Estirable / icono" , MainLauncher = cierto ,
    ConfigurationChanges = ConfigChanges.ScreenSize | ConfigChanges.Orientation)]
    clase pública Actividad principal : global :: Xamarin.Forms.Platform.Android.FormsApplicationActivity
    {
        protected void override OnCreate (Bundle bundle)
        {
            base .OnCreate (bundle);

            global :: Xamarin.Forms.Forms.Init ( esta , Bundle);
            LoadApplication ( nuevo App ());
        }
    }
}

```

La nueva instancia de la Aplicación clase en el Hola espacio de nombres se hace pasar luego a una LoadApplication método definido por FormsApplicationActivity. El atributo establecido en el Actividad principal clase indica que la actividad es *no* recreada cuando la orientación del teléfono cambia (de vertical a horizontal o hacia atrás) o el tamaño cambia la pantalla.

El proyecto Plataforma Windows universal

En el proyecto UWP (o cualquiera de los dos proyectos de Windows), busque primero en el archivo App.xaml.cs escondido debajo del archivo App.xaml en la lista de archivos del proyecto. En el OnLaunched Método verá la llamada a Forms.Init utilizando los argumentos del evento:

```
Xamarin.Forms. formas .init (e);
```

Ahora mira el archivo MainPage.xaml.cs escondido debajo del archivo MainPage.xaml en la lista de archivos del proyecto. Este archivo define la habitual Pagina principal clase, pero en realidad se deriva de una clase Xamarin.Forms especificado como el elemento raíz en el archivo MainPage.xaml. Un recién instanciado Aplicación clase se pasa a la LoadApplication método definido por esta clase de base:

```
espacio de nombres Hello.UWP
{
    público clase parcial sellada Pagina principal
    {
        público Pagina principal()
        {
            esta .InitializeComponent ();

            LoadApplication ( nuevo Hola. Aplicación ());
        }
    }
}
```

¡Nada especial!

Si ha creado una solución Xamarin.Forms en Visual Studio y no desea orientar una o más plataformas, sólo tiene que borrar estos proyectos.

Si más tarde cambia de opinión sobre esos proyectos, o que creó originalmente la solución en Xamarin Studio y desea moverlo a Visual Studio para dirigirse a una de las ventanas plataformas puede agregar nuevos proyectos de plataforma a la solución Xamarin.Forms. En el **Agregar nuevo proyecto** de diálogo, puede crear una API unificada (API no clásico) Xamarin.iOS proyecto seleccionando el proyecto **iOS Universal** tipo y **Aplicación en blanco** modelo. Crear un proyecto Xamarin.Android con el Android **Aplicación en blanco** plantilla, o un proyecto de Windows seleccionando **Universal** bajo la **ventanas** dirección (para un proyecto UWP), o

ventanas o Teléfono windows bajo la windows 8 dirección, y luego Aplicación en blanco.

Para estos nuevos proyectos, puede obtener las referencias correctas y código repetitivo mediante la consulta de los proyectos generados por la plantilla Xamarin.Forms estándar.

En resumen: no hay nada tan especial en una aplicación Xamarin.Forms en comparación con los proyectos-excepto teléfono Xamarin o ventanas normales de las bibliotecas Xamarin.Forms.

PCL o SAP?

Cuando creó la primera **Hola** solución en Visual Studio, que tenía una opción de dos plantillas de aplicación:

- En blanco de la aplicación (Xamarin.Forms Portable)
- Blank App (Xamarin.Forms Compartido)

En Xamarin de estudio, la elección se realiza en un par de botones de radio:

- Utilizar la biblioteca de clases portátil
- El uso de biblioteca compartida

La primera opción crea una biblioteca de clases portátil (PCL), mientras que el segundo crea un proyecto activo compartido (SAP) que consiste solamente en los archivos de código compartido. El original **Hola** solución utiliza la plantilla de PCL. Ahora vamos a crear una segunda solución llamada **HelloSap** con la plantilla de SAP.

Como verá, todo se ve más o menos la misma, excepto que el **HelloSap** proyecto en sí contiene un solo elemento: el archivo App.cs.

Con enfoques tanto el PCL y SAP, el código es compartido entre las cinco aplicaciones, pero en decididamente diferentes maneras: con el método PCL, todo el código común se incluye en una biblioteca de vínculos dinámicos que cada referencias de proyectos de aplicación y se une a en tiempo de ejecución . Con el enfoque de SAP, los archivos de código comunes se incluyen efectivamente con cada uno de los cinco proyectos de aplicación en tiempo de compilación. Por defecto, el SAP tiene un único archivo llamado App.cs, pero efectivamente es como si esta **HelloSap** proyecto no existe y en su lugar había cinco copias diferentes de este archivo en los cinco proyectos de aplicación.

Algunos problemas sutiles (y no tan sutiles) pueden manifestarse con el enfoque de biblioteca compartida:

Los proyectos de iOS y Android tienen acceso a más o menos la misma versión de .NET, pero no es la misma versión de .NET que utilizan los proyectos de Windows. Esto significa que cualquier clases .NET que se accede por el código compartido podrían ser un poco diferente dependiendo de la plataforma. Como descubrirán más adelante en este libro, este es el caso de algún archivo de E / S clases en el System.IO espacio de nombres.

Puede compensar estas diferencias mediante el uso de directivas del preprocesador de C #, especialmente # Si y # elif. En los proyectos generados por la plantilla Xamarin.Forms, los diversos proyectos de aplicación definen los símbolos que se pueden utilizar con estas directivas.

¿Cuáles son estos símbolos?

En Visual Studio, haga clic en el nombre del proyecto en el Explorador de la solución y seleccione Propiedades. A la izquierda de la pantalla de propiedades, seleccione Construir, y buscar la símbolos de compilación condicional campo.

En Xamarin Studio, seleccione un proyecto de aplicación en el Solución lista, invocar las herramientas desplegables

Menú y seleccione Opciones. En el lado izquierdo de la **Opciones del proyecto** diálogo, seleccione **Construir> Compilador**, y buscar la definir **Símbolos** campo.

Estos son los símbolos que se pueden utilizar:

- proyecto IOS: Usted verá el símbolo __ IOS__ (eso es dos guiones antes y después)
- proyecto Android: Usted no verá ningún símbolos definidos para indicar la plataforma, pero el identificador __ ANDROIDE__ se define de todas formas, así como múltiples __ ANDROID_nn__ identificadores, donde nn es cada nivel API de Android compatible.
- proyecto uwp: El símbolo WINDOWS_UWP
- proyecto de Windows: El símbolo WINDOWS_APP
- proyecto de Windows Phone: El símbolo WINDOWS_PHONE_APP

El archivo de código compartido puede incluir bloques como sigue:

```
# Si __IOS__
    // código específico iOS
# elif __ANDROIDE__
    // código específico Android
# elif WINDOWS_UWP
    // código específico de plataforma Windows universal
# elif WINDOWS_APP
    // 8.1 de Windows código específico
# elif WINDOWS_PHONE_APP
    // Windows Phone 8.1 código específico
# terminara si
```

Esto permite que los archivos de código compartido para ejecutar las clases específicas de la plataforma de código de acceso o específicas de la plataforma, incluyendo clases en los proyectos de plataforma individuales. También puede definir sus propios símbolos de compilación condicional si lo desea.

Estas directivas del preprocesador no tienen sentido en un proyecto de biblioteca de clases portátil. El PCL es totalmente independiente de las cinco plataformas, y estos identificadores en los proyectos de plataforma no están presentes cuando se compila el PCL.

El concepto de la PCL originalmente surgió debido a que cada plataforma .NET que utiliza realmente utiliza un subconjunto de .NET algo diferente. Si desea crear una biblioteca que se puede utilizar entre múltiples plataformas .NET, es necesario utilizar sólo las partes comunes de los subconjuntos de .NET.

El PCL está destinada a ayudar al contener código que es utilizable en múltiples (pero específicos) plataformas .NET. En consecuencia, cualquier PCL particular contiene algunas banderas integrados que indican qué plataformas que soporta. A PCL utiliza en una aplicación Xamarin.Forms debe ser compatible con las siguientes plataformas:

- .NET Framework 4.5
- windows 8

- Windows Phone 8.1
- Xamarin.Android
- Xamarin.iOS
- Xamarin.iOS (Classic)

Esto se conoce como perfil PCL 111.

Si necesita un comportamiento específico de la plataforma en el PCL, no puede utilizar las directivas del preprocesador de C# porque aquellos sólo funcionan en tiempo de compilación. Es necesario algo que funciona en tiempo de ejecución, tales como el `Xamarin.Forms Dispositivo clase`. Vas a ver un ejemplo en breve.

El Xamarin.Forms PCL puede acceder a otros PCLs apoyando a las mismas plataformas, pero no puede acceder directamente a las clases definidas en los proyectos de aplicación individuales. Sin embargo, si eso es algo que hay que hacer, y verá un ejemplo en el capítulo 9, "específicas de la plataforma llamadas a la API". Xamarin.Forms proporciona una clase llamada `DependencyService` que le permite acceder a código específico de la plataforma de la PCL de una manera metódica.

La mayoría de los programas de este libro usan el enfoque de PCL. Este es el método recomendado para Xamarin.Forms y es preferido por muchos programadores que han estado trabajando con Xamarin.Forms por un tiempo. Sin embargo, el enfoque de SAP también está apoyado y sin duda tiene sus defensores también. Programas dentro de estas páginas que demuestran el enfoque de SAP siempre contienen las letras **Savia** al final de sus nombres, tales como la **HelloSav** programa.

Pero ¿por qué elegir? Puede tener ambos en la misma solución. Si ha creado una solución Xamarin.Forms con un proyecto activo compartido, se puede añadir un nuevo proyecto PCL a la solución mediante la selección de la **Biblioteca de clases (Xamarin.Forms portátil)** modelo. Los proyectos de aplicaciones pueden acceder tanto al SAP y PCL, y el SAP pueden acceder a la PCL también.

Etiquetas para el texto

Vamos a crear una nueva solución Xamarin.Forms PCL, nombrado **Saludos**, utilizando el mismo proceso descrito anteriormente para crear el **Hola** solución. Esta nueva solución estará estructurado más como un programa Xamarin.Forms típica, lo que significa que va a definir una nueva clase que deriva de `Página de contenido`.

La mayoría de las veces en este libro, todas las clases y estructura definida por un programa tendrá su propio archivo. Esto significa que un nuevo archivo debe ser añadido a la **Saludos** proyecto:

En Visual Studio, puede hacer clic en el **Saludos** proyecto en el **Explorador de la solución** y seleccione **Añadir**
> **Nuevo artículo** en el menú. A la izquierda de la **Agregar ítem** nuevo diálogo, seleccione **Visual C# y CrossPlatform**, y en la zona central, seleccione **ContentPage forma**. (Cuidado con: También hay una **formas ContentView** opción. No recoja que uno!)

En Xamarin de estudio, desde el icono de la herramienta en el **Saludos** proyecto, seleccione **Añadir> Archivo nuevo** desde el

menú. En el lado izquierdo de la Archivo nuevo diálogo, seleccione formas, y en la zona central, seleccione ContentPage forma. (Cuidado con: También hay formas ContentView y ContentPage forma Xaml Opciones. No recoger esos!)

En cualquiera de los casos, dar al nuevo archivo un nombre de GreetingsPage.cs.

El archivo GreetingsPage.cs se inicializará con algo de código esqueleto para una clase llamada SaludaringsPage que deriva de Pagina de contenido. Porque Pagina de contenido está en el Xamarin.Forms espacio de nombres, una utilizando Directiva incluye ese espacio de nombres. La clase se define como pública, pero no tiene que ser debido a que no se puede acceder directamente desde fuera del Saludos proyecto.

Vamos a borrar todo el código en el GreetingsPage constructor y la mayor parte del utilizando directivas, por lo que el archivo se ve algo como esto:

utilizando Sistema;
utilizando Xamarin.Forms;

```
espacio de nombres Saludos
{
    clase pública GreetingsPage : Pagina de contenido
    {
        público GreetingsPage ()
        {

        }
    }
}
```

En el constructor de la GreetingsPage clase, una instancia de una Etiqueta vista, establecer su Texto propiedad, y establecer que Etiqueta instancia a la Contenido propiedad que GreetingsPage hereda de Pagina de contenido:

utilizando Sistema;
utilizando Xamarin.Forms;

```
espacio de nombres Saludos
{
    clase pública GreetingsPage : Pagina de contenido
    {
        público GreetingsPage ()
        {
            Etiqueta etiqueta = nuevo Etiqueta ();
            label.text = "Saludos, Xamarin.Forms!";
            esta .Este contenido ha sido = etiqueta;
        }
    }
}
```

Ahora cambia el Aplicación clase en App.cs para establecer el Página principal propiedad para una instancia de esta SaludaringsPage clase:

utilizando Sistema;
utilizando Xamarin.Forms;

```

espacio de nombres Saludos
{
    clase pública Aplicación : Solicitud
    {
        público App ()
        {
            MainPage = nuevo GreetingsPage ();
        }

        protected void override OnStart ()
        {
            // manipulador cuando se inicia el app
        }

        protected void override OnSleep ()
        {
            // manipulador cuando su aplicación duerme
        }

        protected void override En resumen()
        {
            // manejar cuando sus hojas de vida de aplicaciones
        }
    }
}

```

Es fácil olvidar este paso, y se le desconcertó que su programa parece ignorar por completo la clase de página y todavía dice "Bienvenido a las formas Xamarin!"

Eso está en el `GreetingsPage` clase (y otros similares), donde pasa la mayor parte de su tiempo en la programación Xamarin.Forms temprano. Para algunos de una sola página, programas de interfaz de usuario intensivo, esta clase puede contener el único código de aplicación que tendrá que escribir. Por supuesto, se puede agregar clases adicionales para el proyecto, si los necesita.

En muchos de los programas de ejemplo de una sola página en este libro, la clase que deriva de `Página de contenido` tendrá un nombre que es lo mismo que la aplicación pero con Página anexa. Esta convención de nombres debería ayudar a identificar los listados de código de este libro de sólo el nombre de la clase o el constructor sin ver el archivo completo. En la mayoría de los casos, los fragmentos de código en las páginas de este libro no incluirán la nos-

En g directivas o al espacio de nombres definición.

Muchos programadores Xamarin.Forms prefieren utilizar el estilo de C # 3.0 de la creación de objetos y la inicialización propiedad de sus constructores de página. Usted puede hacer esto por el Etiqueta objeto. Siguiendo el Etiqueta constructor, un par de llaves encierran uno o más valores de propiedades separadas por comas. Aquí es una alternativa (pero funcionalmente equivalente) `GreetingsPage` definición:

```

clase pública GreetingsPage : Página de contenido
{
    público GreetingsPage ()
    {
        Etiqueta etiqueta = nuevo Etiqueta
    }
}

```

```
{
    text = "Saludos, Xamarin.Forms!";
};

esta .Este contenido ha sido = etiqueta;
}

}
```

Este estilo de inicialización propiedad permite la Etiqueta ejemplo que se establece en el Contenido inmueble directamente, de modo que la Etiqueta no requiere un nombre, así:

```
público clase GreetingsPage : Pagina de contenido
{
    público GreetingsPage ()
    {
        content = nuevo Etiqueta
        {
            text = "Saludos, Xamarin.Forms!";
        };
    }
}
```

Para diseños de página más complejos, este estilo de creación de instancias y la inicialización proporciona una mejor analógica visual de la organización de presentaciones y vistas en la página. Sin embargo, no siempre es tan simple como este ejemplo podría indicar si necesita llamar a métodos en estos objetos o conjunto de controladores de eventos.

Cualquiera que sea la forma en que lo hace, si se puede compilar y ejecutar correctamente el programa en el iOS, Android y Windows 10 Las plataformas móviles a ambos un emulador o un dispositivo, esto es lo que verá:



La versión más decepcionante de este Saludos programa es sin duda el iPhone: A partir de iOS 7, una aplicación de una sola página comparte la pantalla con la barra de estado en la parte superior. Cualquier cosa que la aplicación

aparece en la parte superior de su página ocuparán el mismo espacio que la barra de estado a menos que la aplicación compensa.

Este problema desaparece en aplicaciones varias páginas-navegación descrito más adelante en este libro, pero hasta ese momento, aquí hay cuatro maneras (o cinco maneras si está utilizando un SAP) para resolver este problema de inmediato.

Solución 1. Incluir el relleno en la página

los Página clase define una propiedad denominada Relleno que marca un área alrededor del perímetro interior de la página en la que el contenido no puede entrometerse. Los Relleno propiedad es de tipo Espesor, una estructura que define cuatro propiedades con nombre Izquierda, Arriba, Derecha, Abajo. (Es posible que desee memorizar ese orden, porque esa es la orden que definirá las propiedades en el Espesor constructor, así como en XAML.) La Espesor estructura también define constructores para el establecimiento de la misma cantidad de relleno en los cuatro lados o para el establecimiento de la misma cantidad a la izquierda y derecha y en la parte superior e inferior.

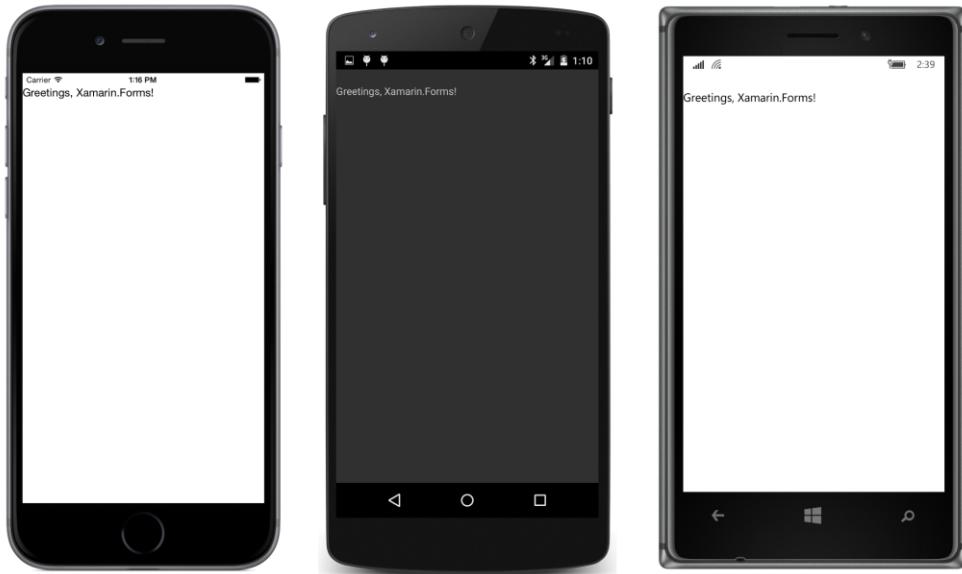
Un poco de investigación en su motor de búsqueda favorito revelará que la barra de estado iOS tiene una altura de 20 (veinte qué? Usted puede preguntar. Veinte pixeles? En realidad, no. Por ahora, sólo pensar en ellos como 20 "unidades". Para gran parte de su programación Xamarin.Forms, que no es necesario preocuparse por tamaños numéricos, pero el capítulo 5, "Tratar con los tamaños", proporcionará una cierta dirección cuando se necesita para llegar hasta el nivel de píxel.)

Puede acomodar la barra de estado, así:

```
espacio de nombres Saludos
{
    clase pública GreetingsPage : Página de contenido
    {
        público GreetingsPage ()
        {
            content = nuevo Etiqueta
            {
                text = "Saludos, Xamarin.Forms!";
            };
        }

        Relleno = nuevo Espesor (0, 20, 0, 0);
    }
}
```

Ahora aparece el saludo de 20 unidades desde la parte superior de la página:



Ajuste de la Relleno propiedad en el Pagina de contenido resuelve el problema del texto sobreescibir la barra de estado iOS, sino que también establece el mismo relleno en el Android y Windows Phone, donde no es necesario. ¿Hay una manera de fijar este relleno sólo en el iPhone?

Solución 2. Incluir el relleno sólo para iOS (SAP solamente)

Una de las ventajas del enfoque compartido Proyecto de Activos (SAP) es que las clases en el proyecto son extensiones de los proyectos de aplicación, lo que puede utilizar directivas de compilación condicional.

Vamos a probar esto a cabo. Vamos a necesitar una nueva solución llamada **GreetingsSap** basado en la plantilla de SAP, y una nueva clase de página en el **GreetingsSap** proyecto denominado **GreetingsSapPage**. Para establecer el Relleno en iOS solamente, esa clase es el siguiente:

```
espacio de nombres GreetingsSap
{
    clase pública GreetingsSapPage : Pagina de contenido
    {
        público GreetingsSapPage ()
        {
            content = nuevo Etiqueta
            {
                text = "Saludos, Xamarin.Forms!"
            };
        }

        # Si __IOS__
        Relleno = nuevo Espesor (0, 20, 0, 0);

        # terminara si
    }
}
```

```
    }  
}  
}
```

Los `# Si` Directiva hace referencia al símbolo de la compilación condicional `__IOS__`, entonces el Relleno propiedad se establece sólo para el proyecto iOS. Los resultados se ven así:



Sin embargo, estos símbolos de compilación condicional afectar sólo a la compilación del programa, por lo que no tendrá ningún efecto en un PCL.

¿Hay una manera para que un proyecto PCL para incluir diferentes Relleno para diferentes plataformas?

Solución 3. Incluir el relleno sólo para iOS (PCL o SAP)

¡Sí! la estática Dispositivo clase incluye varias propiedades y métodos que permiten su código para hacer frente a las diferencias de dispositivos en tiempo de ejecución de una manera muy simple y directo:

- los `Device.OS` propiedad devuelve un miembro de la `TargetPlatform` enumeración: `iOS`, `Android`, `WinPhone`, o `Otro`. `los WinPhone` miembro se refiere a todas las plataformas de Windows y Windows Phone.

- los `Device.Idiom` propiedad devuelve un miembro de la `TargetIdiom` enumeración: `Teléfono`, `Tableta`, `Escritorio`, o `No compatible`.

Puede utilizar estas dos propiedades en `Si` y más declaraciones, o una cambiar y caso bloque, para ejecutar código específico para una plataforma en particular.

Dos métodos llamados `OnPlatform` proporcionar soluciones aún más elegantes:

- El método genérico estático `OnPlatform <T>` tiene tres argumentos de tipo T -el primero para iOS, el segundo para Android, y el tercero para Windows Phone (que abarca todas las plataformas Windows) -y devuelve el argumento a favor de la plataforma de ejecución.
- El método estático `OnPlatform` tiene cuatro argumentos de tipo acción (el delegado función de .NET que no tiene argumentos y devuelve void), también en el orden iOS, Android y Windows Phone, con un cuarto para un defecto, y ejecuta el argumento a favor de la plataforma de ejecución.

En lugar de establecer la misma Relleno propiedad en las tres plataformas, puede restringir la Relleno a sólo el iPhone mediante el uso de la `Device.OnPlatform` método genérico:

```
Relleno = Dispositivo.OnPlatform < Espesor > ( nuevo Espesor (0, 20, 0, 0),
                                                 nuevo Espesor (0),
                                                 nuevo Espesor (0));
```

El primer Espesor argumento es para iOS, el segundo es para Android, y el tercero es para Windows Phone. Explicitamente especificando el tipo de la `Device.OnPlatform` argumentos dentro de los corchetes angulares no es necesario si el compilador puede comprenderlo a partir de los argumentos, por lo que esto funciona así:

```
Relleno = Dispositivo.OnPlatform ( nuevo Espesor (0, 20, 0, 0),
                                   nuevo Espesor (0),
                                   nuevo Espesor (0));
```

O bien, puede tener sólo una Espesor constructor y el uso `Device.OnPlatform` para el segundo argumento:

```
Relleno = nuevo Espesor (0, Dispositivo.OnPlatform (20, 0, 0, 0));
```

Así es como el Relleno por lo general se encuentra en los programas que siguen cuando es necesario. Por supuesto, se puede sustituir algunos otros números de los ceros si quieres un poco de acolchado adicional en la página. A veces un poco de relleno en los lados hace que para una presentación más atractiva.

Sin embargo, si sólo necesita establecer Relleno para iOS, puede utilizar la versión de `Device.OnPlatform` con Acción argumentos. Estos argumentos son nulo de forma predeterminada, por lo que sólo puede establecer la primera para una acción a realizar en iOS:

```
público clase GreetingsPage : Pagina de contenido
{
    público GreetingsPage ()
    {
        content = nuevo Etiqueta
        {
            text = "Saludos, Xamarin.Forms!";
        };

        Dispositivo.OnPlatform (() =>
        {
            Relleno = nuevo Espesor (0, 20, 0, 0);
        });
    }
}
```

Ahora se ejecuta la instrucción para establecer el relleno sólo cuando el programa se ejecuta en iOS. Por supuesto, con sólo que un argumento a `Device.OnPlatform`, que podría ser un poco oscuro para la gente que necesita para leer el código, por lo que es posible que desee incluir el nombre del parámetro que precede al argumento para hacerlo explícito que esta sentencia se ejecuta sólo para iOS:

```
Dispositivo .OnPlatform (iOS: () =>
{
    Relleno = nuevo Espesor (0, 20, 0, 0);
});
```

Nombrando como el argumento de que es una característica introducida en C # 4.0.

los `Device.OnPlatform` método es muy práctico y tiene la ventaja de trabajar en ambos proyectos PCL y SAP. Sin embargo, no se puede acceder a las API dentro de las plataformas individuales. Para que necesitará `DependencyService`, que se discute en el Capítulo 9.

Solución 4. Centro de la etiqueta dentro de la página

El problema con el texto que solapa la barra de estado iOS sólo se produce debido a que la pantalla por defecto del texto está en la esquina superior izquierda. ¿Es posible centrar el texto en la página?

Xamarin.Forms es compatible con una serie de instalaciones para facilitar el diseño sin requerir que el programa para realizar cálculos que implican tamaños y coordenadas. los Ver clase define dos propiedades, nombrados `HorizontalOptions` y `VerticalOptions`, que especifican cómo un punto de vista se ha de colocar en relación con su padre (en este caso el `Pagina` de contenido). Estas dos propiedades son de tipo `LayoutOptions`, una estructura excepcionalmente importante en Xamarin.Forms.

En general, vamos a usar la `LayoutOptions` estructura especificando uno de los ocho campos de sólo lectura estáticos públicos que define que el regreso `LayoutOptions` valores:

- comienzo
- Centrar
- Fin
- Llenar
- StartAndExpand
- CenterAndExpand
- EndAndExpand
- FillAndExpand

Sin embargo, también puede crear una `LayoutOptions` valorate a ti mismo. los `LayoutOptions` estructura también define dos propiedades de la instancia que le permiten crear un valor con estas mismas combinaciones:

- Un Alineación propiedad de tipo `LayoutAlignment`, una enumeración con cuatro miembros:

Comienzo, Centro, Fin, y Llenar.

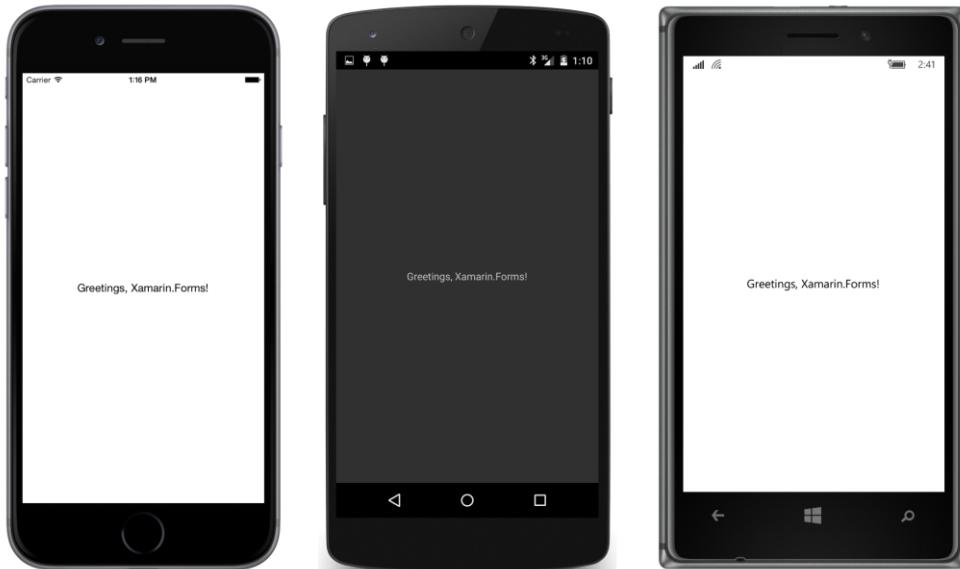
- Un expande propiedad de tipo bool.

Una explicación más completa de todas estas opciones le espera en el Capítulo 4, “Desplazamiento de la pila,” pero por ahora se puede establecer el **HorizontalOptions** y **VerticalOptions** propiedades de la Etiqueta a uno de los campos estáticos definidos por **LayoutOptions** valores. por **HorizontalOptions**, la palabra **comienzo** significa izquierda y **Fin** significa derecho; para **VerticalOptions**, **de inicio** significa la parte superior y **Fin** significa inferior.

Dominar el uso de la **HorizontalOptions** y **VerticalOptions** propiedades es una parte importante de la adquisición de habilidades en el sistema Xamarin.Forms diseño, pero aquí hay un ejemplo sencillo que posiciona el Etiqueta en el centro de la página:

```
p&gt;<code>p&gt;&lt;p&gt;ublic class GreetingsPage : Pagina de contenido<p&gt;{<p&gt;&lt;p&gt;    &lt;p&gt;        &lt;p&gt;            &lt;p&gt;                content = nuevo Etiqueta<p&gt;                {<p&gt;&lt;p&gt;                    text = "Saludos, Xamarin.Forms!"<p&gt;                    HorizontalOptions = LayoutOptions.Centrar,<p&gt;                    VerticalOptions = LayoutOptions.Centrar<p&gt;                };<p&gt;            }<p&gt;        }<p&gt;    }</code>
```

Así es como se ve:



Esta es la versión del **Saludos** programa que se incluye en el código de ejemplo para este capítulo. Se pueden utilizar varias combinaciones de **HorizontalOptions** y **VerticalOptions** para situar el texto en cualquiera de los nueve lugares con respecto a la página.

Solución 5. Centrar el texto dentro de la etiqueta

Los **Etiqueta** tiene la intención de mostrar el texto hasta un párrafo de longitud. A menudo es deseable para controlar la forma de las líneas de texto están alineadas horizontalmente: justificado a la izquierda, a la derecha justificado, o centrada.

Los **Etiqueta** vista define una **HorizontalTextAlignment** propiedad para ese fin y también una **verticalTextAlignment** propiedad para el texto posicionamiento vertical. Ambas propiedades se establecen en un miembro de la **Alineación del texto** enumeración, que cuenta con miembros nombrados **Inicio**, **Centro**, y **Fin** ser lo suficientemente versátil como para el texto que va de derecha a izquierda o de arriba a abajo. Para otros idiomas europeos Inglés y, comienzo significa izquierda o la parte superior y Fin significa derecha o inferior.

Para esta solución definitiva al problema barra de estado iOS, configurar **HorizontalTextAlignment** y **verticalTextAlignment** a **TextAlignment.Center**:

```
clase pública GreetingsPage : Pagina de contenido
{
    público GreetingsPage ()
    {
        content = nuevo Etiqueta
        {
            text = "Saludos, Xamarin.Forms!",
            HorizontalTextAlignment = Alineación del texto .Centrar,
            VerticalTextAlignment = Alineación del texto .Centrar
        };
    }
}
```

Visualmente, el resultado con esta sola línea de texto es el mismo que el entorno **HorizontalOptions** y **VerticalOptions** a **Centrar**, y también se puede utilizar varias combinaciones de estas propiedades para colocar el texto en uno de los nueve lugares diferentes alrededor de la página.

Sin embargo, estas dos técnicas para centrar el texto son en realidad muy diferente, como se verá en el siguiente capítulo.

Capítulo 3

Más profundamente en el texto

A pesar de lo sofisticado que se han convertido en interfaces gráficas de usuario, el texto sigue siendo la columna vertebral de la mayoría de las aplicaciones. Sin embargo, el texto es potencialmente uno de los objetos visuales más complejos, ya que lleva el equipaje de cientos de años de la tipografía. La primera consideración es que el texto debe ser legible. Esto requiere que el texto no sea demasiado pequeña, sin embargo, el texto no debe ser tan grande que acapara mucho espacio en la pantalla.

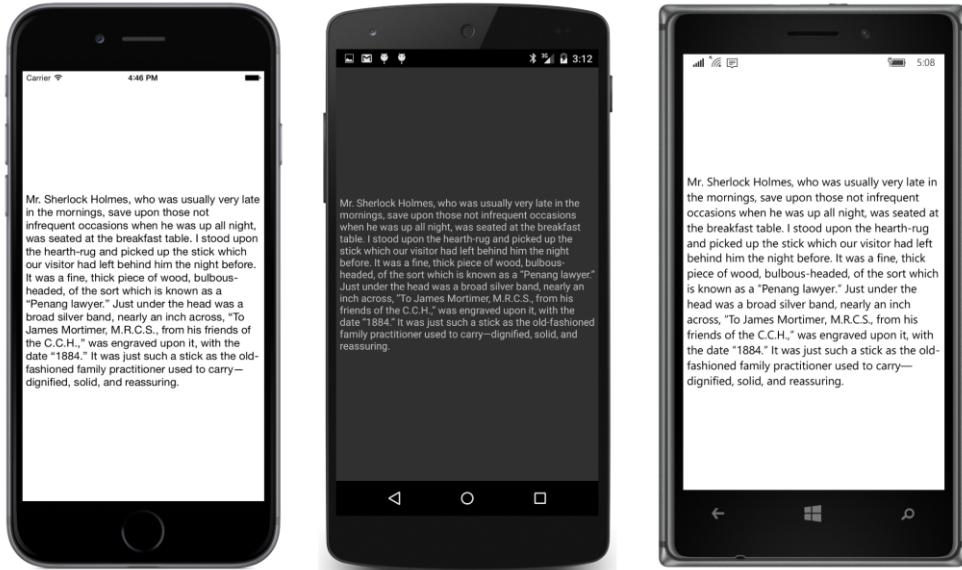
Por estas razones, el tema del texto se continúa en varios capítulos subsiguientes, en particular el capítulo 5, "Tratar con los tamaños." Muy a menudo, los programadores Xamarin.Forms definen las características de fuente de estilos, que son el tema del capítulo 12.

envolver los párrafos

Viendo un párrafo de texto es tan fácil como mostrar una sola línea de texto. Sólo que el texto sea lo suficientemente largo para envolver en varias líneas:

```
público clase BaskervillesPage : Pagina de contenido
{
    público BaskervillesPage ()
    {
        content = nuevo Etiqueta
        {
            VerticalOptions = LayoutOptions .Centrar,
            text =
                "Sherlock Holmes, que era por lo general muy tarde en" +
                "Por las mañanas, salvo a aquellos que no infrecuentan" +
                "Ocasiones cuando estaba despierto toda la noche, se sentó a" +
                "La mesa del desayuno. Yo estaba sobre la chimenea-alfombra" +
                "Y recogió el bastón que nuestro visitante había dejado" +
                "Detrás de él la noche anterior. Era una fina y gruesa" +
                "Trozo de madera, con bulbo-dirigió, del tipo que" +
                "Que se conoce como un abogado \u201CPenang. \u201D Justo" +
                "Debajo de la cabeza era una amplia banda de plata, casi una" +
                "Pulgada de ancho, \u201CTo James Mortimer, Cruz Roja de Myanmar," +
                "De sus amigos de la CCH, \u201D estaba grabado" +
                "Sobre ella, con la fecha \u201C1884. \u201D Era" +
                "Simplemente un palo como la antigua familia" +
                "Practicante utiliza para llevar \u2014dignified, sólido" +
                "Y tranquilizador."
        };
        Relleno = nuevo Espesor (5, Dispositivo .OnPlatform (20, 5, 5), 5, 5);
    }
}
```

Observe el uso de códigos Unicode incrustadas de abierto y cerrado “comillas inteligentes” (\u201c y \u201d) y el guión largo (\u2014). Relleno se ha establecido para 5 unidades alrededor de la página para evitar el texto cabezados contra los bordes de la pantalla, pero la propiedad VerticalOptions ha sido utilizada también para centrar verticalmente todo el párrafo en la página:



Para este párrafo de texto, el establecimiento HorizontalOptions a Inicio, Centro, o Fin en iOS o Windows Phone va a cambiar todo el párrafo horizontalmente hacia la izquierda, centro o derecha. (Android funciona un poco diferente para varias líneas de texto.) El desplazamiento es leve debido a que el ancho del párrafo es el ancho de la línea más larga de texto. Dado que el ajuste de texto se rige por el ancho de página (menos el relleno), el párrafo probablemente ocupa un poco menos ancho que el ancho disponible para el mismo en la página.

Pero el establecimiento de la HorizontalTextAlignment propiedad de la Etiqueta tiene un efecto mucho más profundo: se establece esta propiedad afecta a la alineación de las líneas individuales. Un ajuste de Texto alineado-Center se centrará todas las líneas del párrafo, y TextAlignment.Right todos ellos se alinearán a la derecha. Puedes usar HorizontalOptions además de HorizontalTextAlignment para cambiar todo el párrafo ligeramente hacia el centro o la derecha.

Sin embargo, después de configurar VerticalOptions a Inicio, Centro, o Fin, cualquier ajuste de Vertical- Alineación del texto no tiene efecto.

Etiqueta define una LineBreakMode propiedad que se puede fijar a un miembro de la LineBreakMode enumeración si no desea que el texto se ajuste o para seleccionar las opciones de recorte.

No hay ninguna propiedad para especificar una sangría de primera línea para el párrafo, pero puede agregar uno propio con caracteres de espacio de varios tipos, tales como el espacio largo (Unicode \u2003).

Se pueden visualizar varios párrafos con una sola Etiqueta Ver por finalización de cada párrafo con uno o más caracteres de avance de línea (\ n). Sin embargo, un mejor enfoque es utilizar la cadena devuelta desde el Environment.NewLine propiedad estática. Esta propiedad devuelve "\ n" en iOS y dispositivos Android y "\ r \ n" en todos los dispositivos Windows y Windows Phone. Pero en lugar de incorporar caracteres de salto de línea para crear párrafos, que tiene más sentido usar una separada Etiqueta ver para cada párrafo, como se demostrará en el capítulo 4, "Desplazamiento de la pila."

los Etiqueta clase tiene un montón de formato de flexibilidad. Como veremos en breve, las propiedades definidas por Etiqueta le permiten especificar un tamaño de fuente o el texto en negrita o cursiva, y también se puede especificar el formato de texto diferente dentro de un solo párrafo.

Etiqueta también permite la especificación de color, y un poco de experimentación con el color demostrará la profunda diferencia entre la HorizontalOptions y VerticalOptions propiedades y la horizontalTextAlignment y VerticalTextAlignment propiedades.

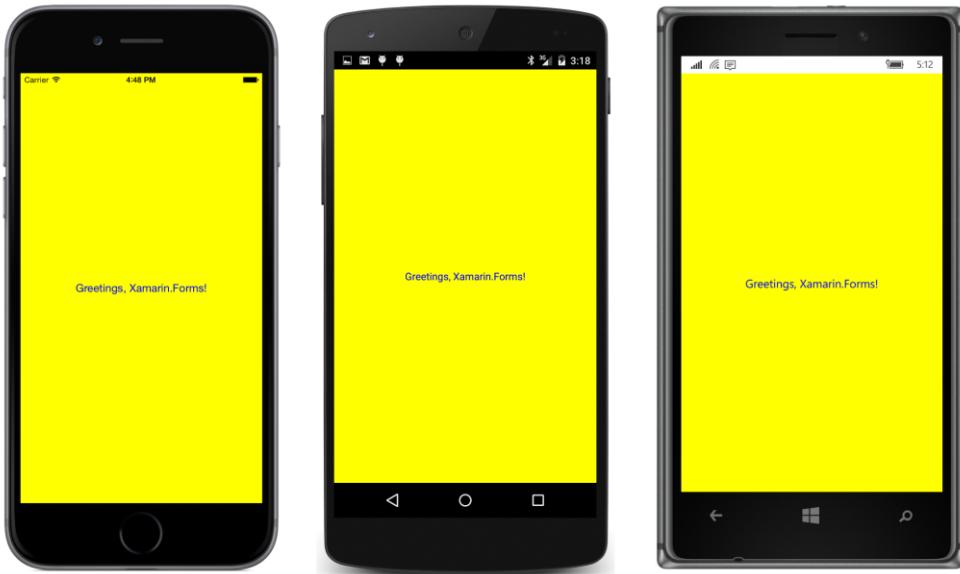
Texto y colores de fondo

Como hemos visto, la Etiqueta vista muestra el texto en una apropiada color para el dispositivo. Puede anular ese comportamiento mediante el establecimiento de dos propiedades, llamada Color de texto y Color de fondo. Etiqueta en sí define Color de texto, pero hereda Color de fondo de VisualElement, Lo que significa que Página y Diseño también tienen una Color de fondo propiedad.

Configura Color de texto y Color de fondo a un valor de tipo Color, que es una estructura que define 17 campos estáticos para la obtención de colores comunes. Usted puede experimentar con estas propiedades con el Saludos programa desde el capítulo anterior. Aquí están dos de estos colores se utilizan en conjunción con HorizontalTextAlignment y VerticalTextAlignment para centrar el texto:

```
público clase GreetingsPage : Pagina de contenido
{
    público GreetingsPage ()
    {
        content = nuevo Etiqueta
        {
            text = "Saludos, Xamarin.Forms!",
            HorizontalTextAlignment = Alineación del texto .Centrar,
            VerticalTextAlignment = Alineación del texto .Centrar,
            BackgroundColor = Color .Amarillo,
            TextColor = Color .Azul
        };
    }
}
```

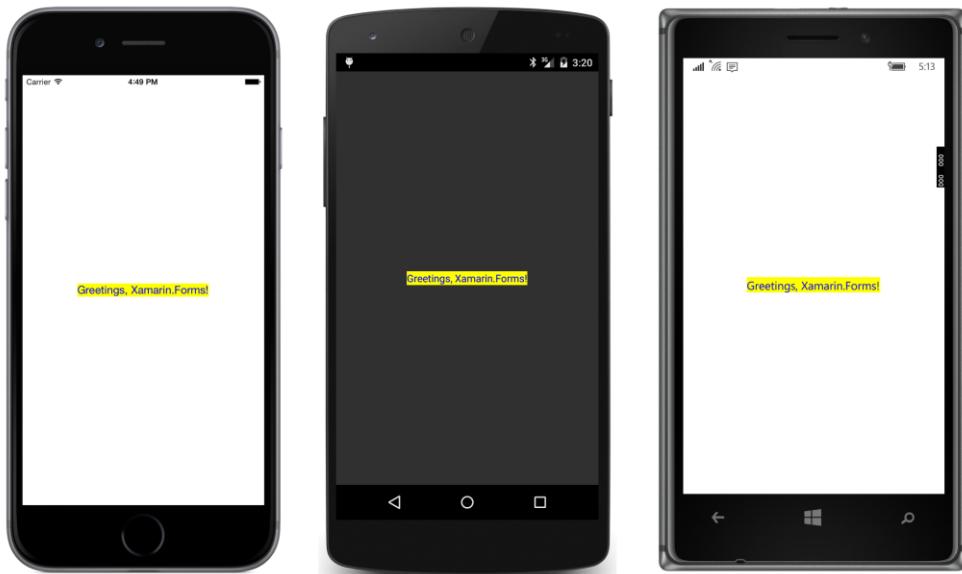
El resultado puede sorprender. Como estas capturas de pantalla ilustran, la Etiqueta en realidad ocupa todo el área de la página (incluyendo debajo de la barra de estado IOS), y el HorizontalTextAlignment y VerticalTextAlignment propiedades colocar el texto dentro de esa área:



Por el contrario, aquí hay un código que los colores del texto de la misma, sino que centra el texto con el **HorizontalOptions** y **VerticalOptions** propiedades:

```
clase pública GreetingsPage : Pagina de contenido
{
    público GreetingsPage ()
    {
        content = nuevo Etiqueta
        {
            text = "Saludos, Xamarin.Forms!",
            HorizontalOptions = LayoutOptions.Centrar,
            VerticalOptions = LayoutOptions.Centrar,
            BackgroundColor = Color.Amarillo,
            TextColor = Color.Azul
        };
    }
}
```

Ahora el Etiqueta sólo ocupa tanto espacio como sea necesario para el texto, y eso es lo que está posicionada en el centro de la página:



El valor por defecto de `HorizontalOptions` y `VerticalOptions` no es `LayoutOptions.Start`, como la apariencia predeterminada del texto podría sugerir. El valor por defecto es su lugar `LayoutOptions.Fill`. Este es el escenario que hace que el Etiqueta para llenar la página. El valor por defecto `HorizontalAlignment` y `VerticalTextAlignment` valor de `TextAlignment.Start` es lo que causó el texto que se coloca en la parte superior izquierda en la primera versión de la **Saludos** programa en el capítulo anterior.

Se pueden combinar varios ajustes de `HorizontalOptions`, `VerticalOptions`, `HorizontalTextAlignment`, y `VerticalTextAlignment` para diferentes efectos.

Usted podría preguntarse: ¿Cuáles son los valores por defecto del Color de texto y Color de fondo propiedades, debido a que los valores por defecto resultan en diferentes colores para las diferentes plataformas?

El valor por defecto de Color de texto y Color de fondo es en realidad un valor de color especial nombrado `Color.Default`, lo que no representa un color real, pero en su lugar se utiliza para hacer referencia a los colores de texto y fondo apropiados para la plataforma en particular.

Vamos a explorar el color con más detalle.

La estructura de color

Internamente, el `Color` almacena los colores de la estructura de dos maneras diferentes:

- Como valores de azul (RGB) de tipo rojo, verde, y doble que van desde 0 a 1. Propiedades de sólo lectura llamado R, G, B, A y segundo exponer a estos valores.

- Como matiz, la saturación y los valores de luminosidad de tipo doble, que también varían de 0 a 1. Estos valores están expuestos con propiedades de sólo lectura denominadas La saturación de color, y Luminosidad.

Los Color estructura también es compatible con un canal alfa para indicar grados de opacidad. Una propiedad de sólo lectura denominada UN expone este valor, que oscila desde 0 para transparente a 1 para opaco.

Todas las propiedades que definen un color son de sólo lectura. En otras palabras, una vez Color Se crea valor, es inmutable.

Se puede crear una Color valor en una de varias maneras. Los tres constructores son los más fáciles:

- nuevo color (doble grayShade)
- nuevo Color (doble r, doble g, doble b)
- nuevo Color (doble r, doble g, doble b, doble a)

Los argumentos pueden variar de 0 a 1. Color también define varios métodos de creación estáticos, incluyendo:

- Color.FromRgb (doble r, doble g, doble b)
- Color.FromRgb (int r, int g, int b)
- Color.FromRgba (doble r, doble g, doble b, doble a)
- Color.FromRgba (int r, int g, int b, int a)
- Color.FromHsla (doble h, dobles s, doble l, doble a)

Los dos métodos estáticos con argumentos enteros asumen que los valores van de 0 a 255, que es la representación habitual de los colores RGB. Internamente, el constructor simplemente divide los valores de número entero por 255,0 para convertir a doble.

¡Cuidado! Se podría pensar que va a crear un color rojo con esta llamada:

```
Color.FromRgb (1, 0, 0)
```

Sin embargo, el compilador de C# asumirá que estos argumentos son enteros. El número entero FromRgb método será invocado, y el primer argumento será dividido por 255,0, con un resultado que es casi cero. Si desea invocar el método que tiene doble argumentos, sean explícitos:

```
Color.FromRgb (1.0, 0, 0)
```

Color también define métodos de creación estáticos para una pila de discos uint formato y un formato hexadecimal en una cadena, pero éstos se utilizan con menos frecuencia.

Los Color También define la estructura 17 public static campos de sólo lectura de tipo Color. En la tabla siguiente, el número entero RGB valores que el Color estructura utiliza internamente para definir estos campos se muestran junto con la correspondiente La saturación de color, y Luminosidad valores, algo redondeados por motivos de claridad:

campos de color	Color	rojo	Verde	Azul	Matiz	Saturación	Luminosidad
Blanco		255	255	255	0	0	1.00
Plata		192	192	192	0	0	0.75
gris		128	128	128	0	0	0.50
Negro		0	0	0	0	0	0
rojo		255	0	0	1.00	1	0.50
Granate		128	0	0	1.00	1	0.25
Amarillo		255	255	0	0.17	1	0.50
Aceituna		128	128	0	0.17	1	0.25
Lima		0	255	0	0.33	1	0.50
Verde		0	128	0	0.33	1	0.25
Agua		0	255	255	0.50	1	0.50
Teal		0	128	128	0.50	1	0.25
Azul		0	0	255	0.67	1	0.50
Armada		0	0	128	0.67	1	0.25
Rosado		255	102	255	0.83	1	0.70
Fucsia		255	0	255	0.83	1	0.50
Púrpura		128	0	128	0.83	1	0.25

Con la excepción de Rosado, Usted puede reconocer estos como los nombres de colores soportadas en HTML. Un public static 18a campo de sólo lectura se llama **Transparente**, que tiene R, GRAMO, SEGUNDO, y UN propiedades ajustan a cero.

Cuando las personas se les da la oportunidad de formular de forma interactiva un color, el modelo de color HSL es a menudo más intuitivo que RGB. Los Matiz ciclos a través de los colores del espectro visible (y el arco iris) comenzando con rojo a 0, verde a 0,33, azul a 0,67, y de vuelta a rojo en 1.

Los Saturación indica el grado de la tonalidad en el color, que van desde 0, lo cual no es en absoluto el tono y los resultados en un tono gris, a 1 para la saturación completa.

Los Luminosidad es una medida de la claridad, que van desde 0 para el negro a 1 para el blanco.

Programas de selección de color en el capítulo 15, "La interfaz interactiva," le permiten explorar los modelos RGB y HSL más interactiva.

Los Color estructura incluye varios métodos de instancia interesantes que permiten la creación de nuevos colores que son modificaciones de colores existentes:

- AddLuminosity (doble delta)
- MultiplyAlpha (doble alfa)
- WithHue (doble newHue)
- WithLuminosity (doble newLuminosity)
- WithSaturation (doble newSaturation)

Finalmente, Color define dos propiedades de sólo lectura estáticas especiales de tipo Color:

- Color.Default
- Color.Accent

los `Color.Default` propiedad se utiliza ampliamente en Xamarin.Forms para definir el color predeterminado de puntos de vista. Los `VisualElement` clase inicializa su `Color de fondo` propiedad a `Color.Default`, y el `Etiqueta` clase inicializa su `Color de texto` propiedad como `Color.Default`.

Sin embargo, `Color.Default` es un `Color` con su valor R, GRAMO, SEGUNDO, y UN propiedades ajustan a -1, lo que significa que es un valor especial "simulacro" que no significa nada en sí mismo, sino que indica que el valor real es específica de la plataforma.

por `Etiqueta` y `Página de contenido` (y la mayoría de las clases que se derivan de `VisualElement`), el `Color de fondo` ajuste de `Color.Default` significa transparente. El color de fondo que se ve en la pantalla es el color de fondo de la página. Los `Color de fondo` propiedad de la página tiene una configuración predeterminada de `Color.Default`, pero ese valor significa algo diferente en las distintas plataformas. El significado de `Color.Default` Para el `Color de texto` propiedad de `Etiqueta` es también dependiente del dispositivo.

Aquí están los esquemas de color por defecto que implica la `Color de fondo` de la página y la `Color de texto` del `Etiqueta`:

Plataforma	Esquema de colores
iOS	Texto oscuro sobre un fondo claro
Android	texto claro sobre un fondo oscuro
UWP	Texto oscuro sobre un fondo claro
de windows 8.1	texto claro sobre un fondo oscuro
Windows Phone 8.1	texto claro sobre un fondo oscuro

En Android, Windows y los dispositivos Windows Phone, puede cambiar este esquema de color para su aplicación. Consulte la siguiente sección.

Usted tiene un par de posibles estrategias para trabajar con los colores: Usted puede optar por hacer sus Xamarin.Forms programación de una manera muy independiente de la plataforma y evitar hacer ninguna suposición acerca de la combinación de colores por defecto de cualquier teléfono. O bien, puede usar su conocimiento acerca de los esquemas de color de las diversas plataformas y el uso `Device.OnPlatform` para especificar los colores específicos de la plataforma.

Pero no se trata de simplemente ignorar todos los valores predeterminados de la plataforma y explícitamente establecido todos los colores de la aplicación a su propio esquema de color. Esto probablemente no funcionará tan bien como usted espera debido a que muchas vistas utilizan otros colores que se relacionan con el tema del color del sistema operativo, pero que no están expuestos a través de propiedades Xamarin.Forms.

Una opción sencilla es utilizar el `Color.Accent` viviendas en un color de texto alternativo. En las plataformas iPhone y Android, este es un color que es visible contra el fondo por defecto, pero no es el color de texto predeterminado. En las plataformas Windows, que es un color seleccionado por el usuario como parte del tema de color.

Puede hacer semitransparente de texto estableciendo `Color de texto` a una `Color` valor con una UN propiedad inferior a 1. Sin embargo, si quieres una versión semitransparente del color de texto predeterminado, utilice el Opacidad propiedad de la `Etiqueta` en lugar. Esta propiedad se define por la `VisualElement` clase y tiene un valor predeterminado de 1. Establecer que a valores menores que 1 para diferentes grados de transparencia.

Cambiar el esquema de color de la aplicación

Cuando la orientación de su aplicación para Android, Windows y Windows Phone, es posible cambiar el esquema de color para la aplicación. En este caso, la configuración de Color.Default Para el Espalda-groundColor del Pagina de contenido y el Color de texto propiedad de la Etiqueta tendrá diferentes significados.

Hay varias formas de establecer esquemas de color en Android, pero la más sencilla requiere sólo un único valor en el archivo `AndroidManifest.xml` en el atributo **propiedades** carpeta del proyecto Android. Ese archivo normalmente se ve así:

```
< manifiesto xmlns: android = " http://schemas.android.com/apk/res/android " >
    < use-SDK android: minSdkVersion = " 15 " />
    < solicitud >
        </ solicitud >
</ manifiesto >
```

Añadir el siguiente atributo a la solicitud etiqueta:

```
< manifiesto xmlns: android = " http://schemas.android.com/apk/res/android " >
    < usa-SDK android: minSdkVersion = " 15 " />
    < solicitud android: el tema = " @ Estilo / android: Theme.Holo.Light " >
        </ solicitud >
</ manifiesto >
```

Ahora su aplicación para Android mostrará el texto oscuro sobre un fondo claro.

Para los tres proyectos de Windows y Windows Phone, tendrá que cambiar el archivo `App.xaml` situado en el proyecto en particular.

En el **UWP** proyecto, el archivo `App.xaml` por defecto es el siguiente:

```
< Solicitud
    X : Clase = "Baskervilles.UWP.App"
    xmlns = "Http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns : X = "Http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns : local = "Usando: Baskervilles.UWP"
    RequestedTheme = "Light">

</ Solicitud >
```

Ese `RequestedTheme` atributo es el que da la aplicación uwp un esquema de color del texto oscuro sobre un fondo claro.

Cambiarlo a Oscuro de texto claro sobre un fondo oscuro. Eliminar el Pedido-

Tema atribuir por completo para permitir la configuración del usuario para determinar la combinación de colores.

El archivo `App.xaml` para los proyectos de Windows Phone 8.1 y Windows 8.1 es similar, pero el `Solicitud-edTheme` atributo no se incluye por defecto. Aquí está el archivo en el `App.xaml WinPhone` proyecto:

```
< Solicitud
    X : Clase = "Baskervilles.WinPhone.App"
```

```
xmlns = "Http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns : X = "Http://schemas.microsoft.com/winfx/2006/xaml"
xmlns : local = "Usando: Baskervilles.WinPhone">
```

</ Solicitud >

Por defecto, la combinación de colores está determinada por la configuración del usuario. Puede incluir una RequestedTheme atribuir y ponerlo a Ligero o Oscuro para anular la preferencia del usuario y tomar el control de la combinación de colores.

Configurando RequestedTheme en sus proyectos de Windows Phone y Windows, la aplicación debe tener un conocimiento completo de los esquemas de color subyacentes en todas las plataformas.

tamaños y atributos de fuente

Por defecto, el Etiqueta utiliza una fuente de sistema definido por cada plataforma, pero Etiqueta también define varias propiedades que se pueden utilizar para cambiar esta fuente. Etiqueta es uno de sólo dos clases con estas propiedades relacionadas con la fuente; Botón es el otro.

Las propiedades que le permiten cambiar esta fuente son:

- Familia tipográfica de tipo cuerda
- Tamaño de fuente de tipo doble
- FontAttributes de tipo FontAttributes, una enumeración con tres miembros: Ninguno, Negrita, y Itálico.

También hay una Fuente propiedad y que corresponde Fuente estructura, pero esto es obsoleto y no debe utilizarse.

El más difícil de usar estos para es Familia tipográfica. En teoría se puede establecer a un nombre de familia de fuentes tales como "Times New Roman", pero sólo funcionará si esa familia de la fuente en particular es compatible con la plataforma en particular. Por esta razón, es probable que utilice Familia tipográfica en conexión con Device.OnPlatform, y usted necesita saber los nombres de la familia de fuentes compatibles de cada plataforma.

Los Tamaño de fuente la propiedad es un poco incómodo también. Es necesario un número que indica aproximadamente la altura de la fuente, pero lo que los números se debe utilizar? Este es un tema espinoso, y por esa razón, se relegó al Capítulo 5, "Tratar con los tamaños", cuando las herramientas para recoger un buen tamaño de la fuente estará disponible.

Hasta entonces, sin embargo, el Dispositivo clase ayuda a cabo con un método estático llamado GetNamedSize. Este método requiere un miembro de la NamedSize enumeración:

- Defecto
- Micro

- Pequeña
- Medio
- Grande

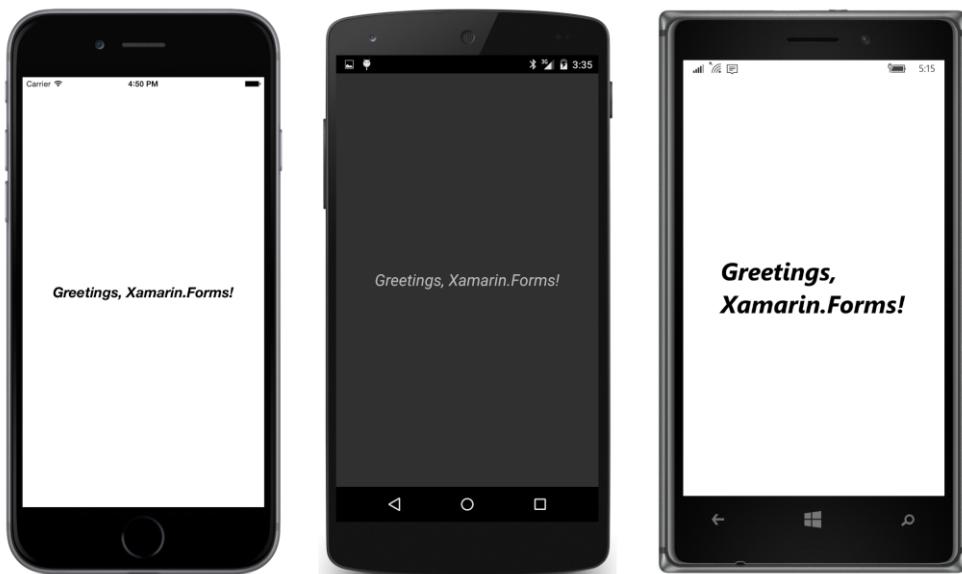
`GetNamedSize` También requiere que el tipo de la clase que está dimensionado con este tamaño de fuente, y que será un argumento `typeof(Label)` o `typeof(Button)`. También puede utilizar una instancia de `Etiqueta` o `Botón` si en lugar de la `Tipo`, pero esta opción es a menudo menos conveniente.

Como se verá más adelante en este capítulo, el `NamedSize.Medium` miembro no devuelve necesariamente del mismo tamaño que `NamedSize.Default`.

`FontAttributes` es la menos complicada de las tres propiedades relacionadas con la fuente de usar. Puede especificar Negrita o Itálico o ambos, como este pequeño fragmento de código (adaptado de la **Saludos** programa desde el capítulo anterior) demuestra:

```
clase GreetingsPage : Pagina de contenido
{
    público GreetingsPage ()
    {
        content = nuevo Etiqueta
        {
            text = "Saludos, Xamarín.Forms!",
            HorizontalOptions = LayoutOptions.Centrar,
            VerticalOptions = LayoutOptions.Centrar,
            Tamaño de Letra = Dispositivo.GetNamedSize ( NamedSize.Grande, tipo de ( Etiqueta )),
            FontAttributes = FontAttributes.bold | FontAttributes.Itálico
        };
    }
}
```

Aquí está en las tres plataformas:



La pantalla móvil de Windows 10 no es bastante lo suficientemente amplia como para mostrar el texto en una sola línea.

El texto con formato

Como hemos visto, **Etiqueta** tiene un **Texto** propiedad que se puede establecer en una cadena. Pero **Etiqueta** También tiene una alternativa **FormattedText** propiedad que construye un párrafo con formato uniforme.

Los **FormattedText** propiedad es de tipo **FormattedString**, que tiene una **Palmas** propiedad de tipo **IList**, una colección de **Lapso** objetos. Cada **Lapso** objeto es un trozo de manera uniforme con formato de texto que se rige por seis propiedades:

- Texto
- Familia tipográfica
- Tamaño de fuente
- FontAttributes
- Color de primer plano
- Color de fondo

Aquí está una manera de crear una instancia de una **FormattedString** objeto y luego añadir **Lapso** instancias a su **Palmas** propiedad de colección:

```
clase pública VariableFormattedTextPage : Pagina de contenido
{

```

```


publico VariableFormattedTextPage ()



{



FormattedString formatedString = nuevo FormattedString ();



    formatedString.Spans.Add ( nuevo Lapso



    {



        text = "YO "



    });



    formatedString.Spans.Add ( nuevo Lapso



    {



        text = "amor",



        Tamaño de Letra = Dispositivo .GetNamedSize ( NamedSize .Grande, tipo de ( Etiqueta )),



        FontAttributes = FontAttributes .Negrita



    });



    formatedString.Spans.Add ( nuevo Lapso



    {



        text = "Xamarín.Forms!"



    });



    content = nuevo Etiqueta



    {



        FormattedText = formatedString,



        HorizontalOptions = LayoutOptions .Centrar,



        VerticalOptions = LayoutOptions .Centrar,



        Tamaño de Letra = Dispositivo .GetNamedSize ( NamedSize .Grande, tipo de ( Etiqueta ))



    };



}


}

```

Como cada Lapso se crea, se pasa directamente a la Añadir método de la palmos colección. Observe que el Etiqueta se le da una Tamaño de fuente de NamedSize.Large, y el Lapso con el Negrita ajuste también se da explícitamente que mismo tamaño. Cuando un Lapso se le da una FontAttributes ajuste, que no hereda el

Tamaño de fuente ajuste de la Etiqueta.

Alternativamente, es posible inicializar el contenido de la palmos colección, siguiendo con un par de llaves. Dentro de estas llaves, la Lapso objetos se instancian. Porque no se requieren llamadas a métodos, la totalidad FormattedString inicialización puede ocurrir dentro de la Etiqueta inicialización:

```


publico clase VariableFormattedTextPage : Pagina de contenido



{



publico VariableFormattedTextPage ()



    {



        content = nuevo Etiqueta



        {



            FormattedText = nuevo FormattedString



            {



                Los tramos =



                {



nuevo Lapso



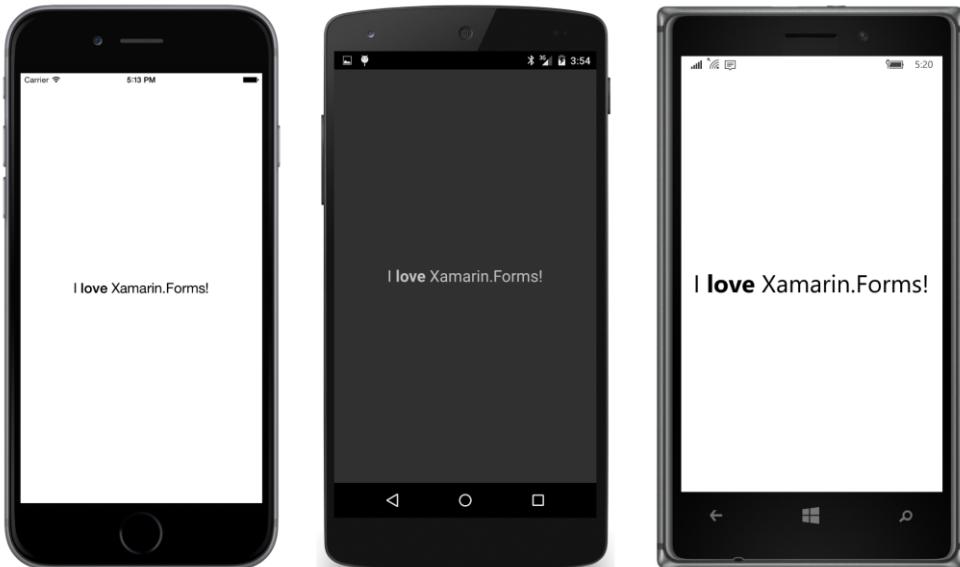
                    {



                        text = "YO "


```

Esta es la versión del programa que se verá en la colección de código de ejemplo para este capítulo. Independientemente del método que se utiliza, esto es lo que parece:



También puede utilizar el FormattedText propiedad para incrustar palabras cursiva o negrita dentro de un párrafo entero, como el **VariableFormattedText** que el programa demuestra:

```
clase pública VariableFormattedParagraphPage : Página de contenido  
{  
    público VariableFormattedParagraphPage ()  
}
```

```

content = nuevo Etiqueta
{
    FormattedText = nuevo FormattedString
    {
        Los tramos =
        {
            nuevo Lapso
            {
                text = "\U2003There había nada por lo que"
            },
            nuevo Lapso
            {
                text = "muy",
                FontAttributes = FontAttributes .Itálico
            },
            nuevo Lapso
            {
                text = "Extraordinario en esto, ni tampoco Alice" +
                    "Creo que lo que"
            },
            nuevo Lapso
            {
                text = "muy",
                FontAttributes = FontAttributes .Itálico
            },
            nuevo Lapso
            {
                text = "Gran parte del camino a escuchar el" +
                    "Conejo se decla a sí mismo \u2018Oh" +
                    "Querida! ¡Dios mío! Me será demasiado tarde!" +
                    "\U2019 (cuando ella lo pensó" +
                    "Después, se le ocurrió que" +
                    "Que debería haber preguntado en este" +
                    "Pero en ese momento le pareció lo más" +
                    "Natural); pero, cuando el Conejo realidad"
            },
            nuevo Lapso
            {
                text = "Sacó un reloj de su bolsillo del chaleco",
                FontAttributes = FontAttributes .Itálico
            },
            nuevo Lapso
            {
                text = "Y lo miró, y luego se apresuró en adelante," +
                    "Alice empezó a ponerse en pie, porque destellaba" +
                    "A través de su mente que ella nunca antes había tenido" +
                    "Visto un conejo, ya sea con un waistcoat" +
                    "Bolsillo, o un reloj para sacar de ella" +
                    "Y, muerta de curiosidad, corrió" +
                    "A través del campo después de ella, y se fue afuera" +
                    "A tiempo para ver cómo se precipitaba en una gran" +
                    "Conejo-mantener bajo el seto."
            }
        }
    }
}

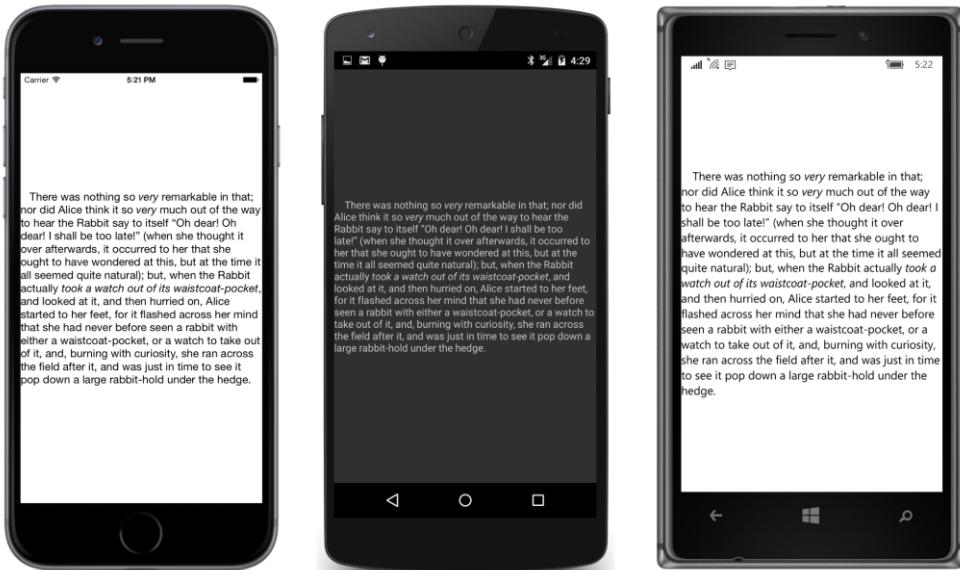
```

```

    },
    HorizontalOptions = LayoutOptions.Centrar,
    VerticalOptions = LayoutOptions.Centrar
);
}
}
}

```

El párrafo comienza con un espacio largo (Unicode \ u2003) y contiene los llamados comillas tipográficas (\ u201C y \ u201D), y varias palabras están en cursiva:



Se puede persuadir a un solo Etiqueta para mostrar varias líneas o párrafos con la inserción de caracteres de línea de endof.

Esto se demuestra en el **NamedFontSize** programa. Múltiple Lapsos objetos se añaden a una **FormattedString** objeto en una para cada lazo.

Cada Lazo objeto utiliza un diferente

NamedFont valor y también muestra el tamaño real de regresar de `Device.GetNamedSize`:

```

clase pública NamedFontSizePage : Pagina de contenido
{
    público NamedFontSizePage ()
    {
        FormattedString formatedString = nuevo FormattedString ();
        NamedSize [] NamedSizes =
        {
            NamedSize .Defecto, NamedSize .Micro, NamedSize .Pequeña,
            NamedSize .Medio, NamedSize .Grande
        };

        para cada ( NamedSize namedSize en namedSizes )
        {
            doble fontSize = Dispositivo .GetNamedSize (namedSize, tipo de ( Etiqueta ));

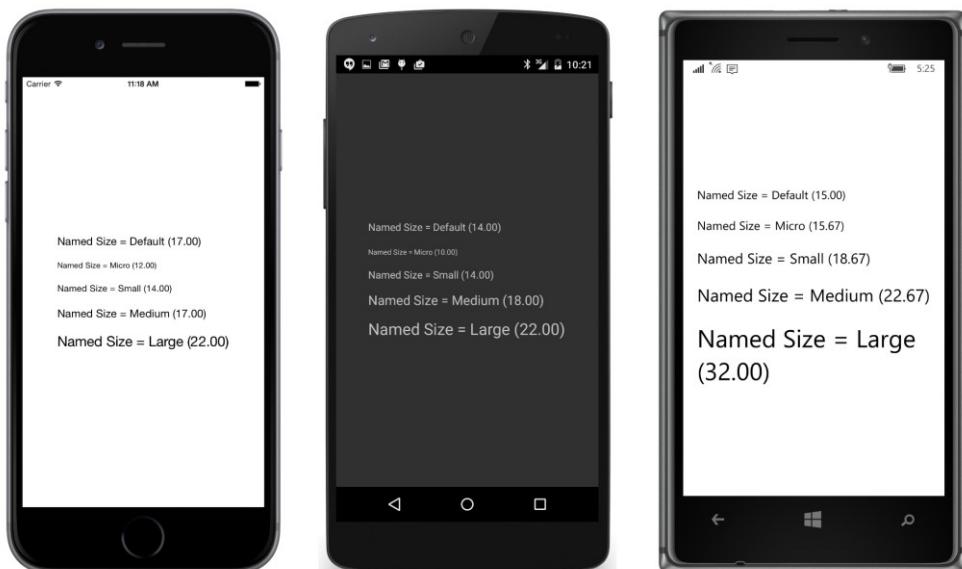
```

```
formattedString.Spans.Add ( nuevo Lapso
{
    text = Cuerda .Formato( "Tamaño Named = {0} ({1}: F2)" ,
                           namedSize, fontSize),
    FontSize = fontSize
});

Si (NamedSize! = NamedSizes.Last ())
{
    formattedString.Spans.Add ( nuevo Lapso
    {
        text = Ambiente .NewLine + Ambiente .Nueva línea
    });
}

content = nuevo Etiqueta
{
    FormattedText = formattedString,
    HorizontalOptions = LayoutOptions .Centrar,
    VerticalOptions = LayoutOptions .Centrar
};
}
}
```

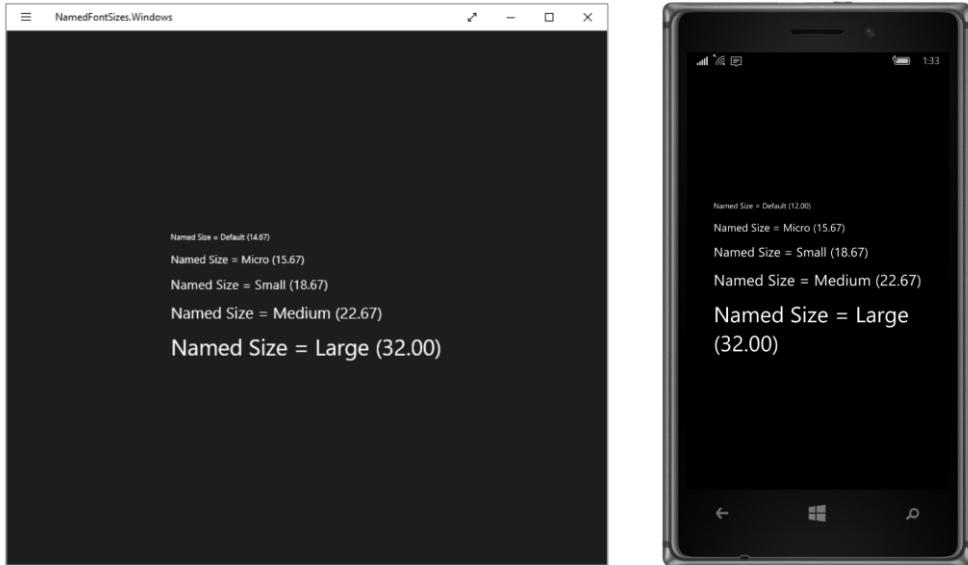
Tenga en cuenta que una por separado `Lapso` contiene las dos cadenas específicas de la plataforma de fin de línea para el espacio de las líneas individuales. Esto asegura que el espaciado de línea se basa en el tamaño de fuente predeterminado en lugar de simplemente el tamaño de fuente que se muestra:



No se trata de tamaños de pixel! Al igual que con la altura de la barra de estado iOS, lo mejor es hacer referencia a estos tamaños vagamente como una especie de "unidades". Algunos mayor claridad viene en el capítulo 5.

los Defecto tamaño se elige generalmente por el sistema operativo, pero los otros tamaños fueron elegidos por los desarrolladores Xamarin.Forms. En iOS, Defecto es lo mismo que Medio, pero en Android Defecto es lo mismo que Pequeña, y en Windows 10 móvil, Defecto es más pequeña que Micro.

Los tamaños en el iPad y Windows 10 son los mismos que el iPhone y Windows Mobile 10, respectivamente. Sin embargo, los tamaños en el teléfono de Windows 8.1 y Windows 8.1 plataformas muestran más de discrepancia:



Por supuesto, el uso de múltiples Lapsos objetos en un solo Etiqueta no es una buena manera de procesar varios párrafos de texto. Por otra parte, el texto a menudo tiene tantos párrafos que hay que desplazar. Este es el trabajo para el siguiente capítulo y su exploración de StackLayout y ScrollView.

Capítulo 4

Hojar en la pila

Si usted es como la mayoría de los programadores, tan pronto como vio que la lista de estática Color propiedades en el capítulo anterior, que quería escribir un programa para mostrar a todos ellos, tal vez usando el Texto propiedad de Etiqueta para identificar el color, y la Color de texto propiedad para mostrar el color real.

Aunque se puede hacer esto con una sola Etiqueta usando un FormattedString objeto, que es mucho más fácil con múltiples Etiqueta objetos. debido a múltiples Etiqueta los objetos están involucrados, este trabajo también requiere una cierta manera de mostrar toda la Etiqueta objetos en la pantalla.

los Pagina de contenido clase define una Contenido propiedad de tipo Ver que se puede establecer a un objeto- pero sólo un objeto. La presentación de varias vistas requiere ajuste Contenido a una instancia de una clase que puede tener varios hijos de Tipo Ver. una clase de este tipo es Disposición <T>, que define una Niños propiedad de tipo IList <T>.

los Disposición <T> clase es abstracta, sino cuatro clases derivan de Disposición <Ver>, una clase que puede tener varios hijos de Tipo Ver. En orden alfabético, estas cuatro clases son:

- AbsoluteLayout
- Cuadricula
- Disposición relativa
- StackLayout

Cada una de las organiza sus hijos de una manera característica. Este capítulo se centra en StackLayout.

Las pilas de vistas

los StackLayout clase organiza sus hijos en una pila. En él se definen sólo dos propiedades en su propia:

- Orientación de tipo StackOrientation, una enumeración con dos miembros: Vertical (el valor por defecto) y Horizontal.
- Espaciado de tipo doble, inicializado a 6,0.

StackLayout parece ideal para el trabajo de los colores de listado. Se puede utilizar el Añadir método definido por IList <T> añadir a los niños a la Niños cobro de una StackLayout ejemplo. Aquí hay algo de código que crea múltiples Etiqueta objetos a partir de dos matrices y luego se añade cada uno Etiqueta al Niños cobro de una StackLayout:

```

    {
        Color .Blanco, Color .Plata, Color .Gris, Color .Negro, Color .Rojo,
        Color .Granate, Color .Amarillo, Color .Aceituna, Color .Lima, Color .Verde,
        Color .Agua, Color .Teal, Color .Azul, Color .Armedia, Color .Rosado,
        Color .Fucsia, Color .Púrpura
    };

    cuerda [] ColorNames =
    {
        "Blanco", "Plata", "Gris", "Negro",
        "Granate", "Amarillo", "Aceituna", "Lima", "Verde",
        "Agua", "Teal", "Azul", "Armedia", "Rosado",
        "Fucsia", "Púrpura"
    };

    StackLayout stackLayout = nuevo StackLayout ();

    para ( En t i = 0; i < colors.Length; i++)
    {
        Etiqueta etiqueta = nuevo Etiqueta
        {
            Text = colorNames [i],
            TextColor = colores [i],
            Tamaño de Letra = Dispositivo .GetNamedSize ( NamedSize .Grande, tipo de ( Etiquetas ))
        };
        stackLayout.Children.Add (etiqueta);
    }
}

```

los StackLayout objeto puede ser entonces ajustado a la Contenido propiedad de la página.

Sin embargo, la técnica de utilizar matrices paralelas es bastante peligroso. ¿Y si están fuera de sincronía o tienen un número diferente de elementos? Un mejor enfoque es mantener el color y el nombre juntos, tal vez en una pequeña estructura con Color y Nombre campos, o como una matriz de Tupla <color, cadena> valores, o como un tipo anónimo, como se demuestra en el **ColorLoop** programa:

```

clase ColorLoopPage : Pagina de contenido
{
    público ColorLoopPage ()
    {
        var colores = nuevo []
        {
            nuevo {Value = Color .white, name = "Blanco"},
            nuevo {Value = Color .Silver, name = "Plata"},
            nuevo {Value = Color .gray, name = "Gris"},
            nuevo {Value = Color .Black, name = "Negro"},
            nuevo {Value = Color .Red, name = "Rojo"},
            nuevo {Value = Color .Maroon, name = "Granate"},
            nuevo {Value = Color .yellow, name = "Amarillo"},
            nuevo {Value = Color .Olive, name = "Aceituna"},
            nuevo {Value = Color .Lime, name = "Lima"},
            nuevo {Value = Color .Green, name = "Verde"},
            nuevo {Value = Color .Aqua, name = "Agua"},
            nuevo {Value = Color .Teal, name = "Teal"},
        }
    }
}

```

```

nuevo {Value = Color.Blue, name = "Azul"},  

nuevo {Value = Color.Navy, name = "Armada"},  

nuevo {Value = Color.pink, name = "Rosado"},  

nuevo {Value = Color.Fuchsia, name = "Fucsia"},  

nuevo {Value = Color.purple, name = "Púrpura"}  

};  
  

StackLayout stackLayout = nuevo StackLayout();  
  

para cada ( var color en colores)  

{  

    stackLayout.Children.Add(  

        nuevo Etiqueta  

        {  

            Text = color.name,  

            TextColor = color.value,  

            Tamaño de Letra = Dispositivo.GetNamedSize( NamedSize.Grande, tipo de ( Etiqueta ))  

        });
}  

}  
  

Relleno = nuevo Espesor(5, Dispositivo.OnPlatform(20, 5, 5), 5, 5);  

Content = stackLayout;  

}  

}

```

O puede inicializar el Niños propiedad de StackLayout con una colección explícita de puntos de vista (similar a la forma en que el palmos cobro de una FormattedString objeto se ha inicializado en el capítulo anterior). Los colorlist programa establece el Contenido propiedad de la página para una StackLayout objeto, que entonces tiene su Niños propiedad inicializado con 17 Etiqueta puntos de vista:

```

clase ColorListPage : Pagina de contenido  

{  

    público ColorListPage ()  

    {  

        Relleno = nuevo Espesor(5, Dispositivo.OnPlatform(20, 5, 5), 5, 5);  

        doble fontSize = Dispositivo.GetNamedSize( NamedSize.Grande, tipo de ( Etiqueta ));  

        content = nuevo StackLayout  

        {  

            Los niños =  

            {  

                nuevo Etiqueta  

                {  

                    text = "Blanco",  

                    TextColor = Color.Blanco,  

                    FontSize = fontSize
                },  

                nuevo Etiqueta  

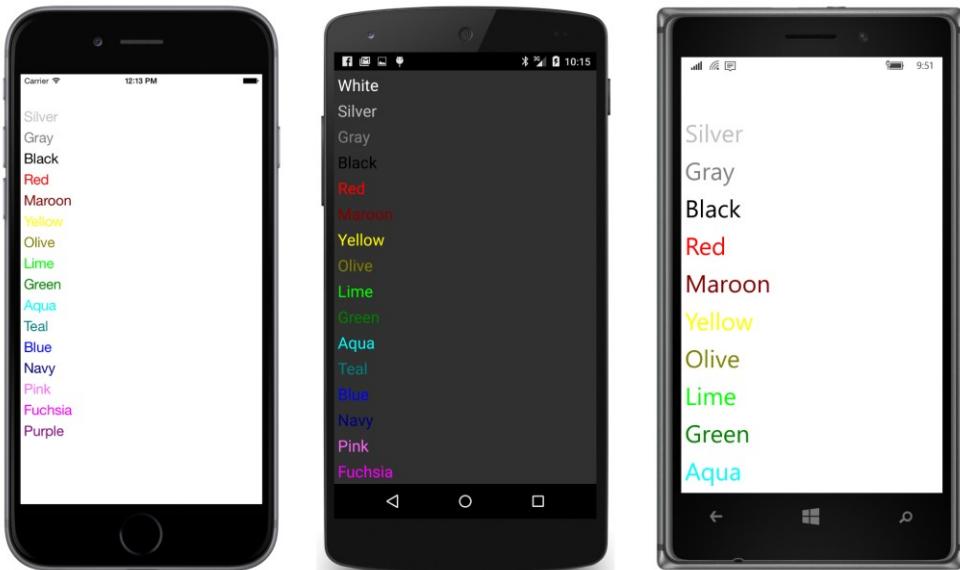
                {  

                    text = "Plata",  

                    TextColor = Color.Plata,
                    FontSize = fontSize
                },
            }
        }
    }
}
```

```
nuevo Etiqueta
{
    text = "Fucsia",
    TextColor = Color .Fucsia,
    FontSize = fontSize
},
nuevo Etiqueta
{
    text = "Púrpura",
    TextColor = Color .Púrpura,
    FontSize = fontSize
}
}
}
}
```

Nos es necesario para ver el código para todos los 17 niños para obtener la idea! Independientemente de cómo se llena el



Obviamente, esto no es óptima. Algunos colores no son visibles en absoluto, y algunos de ellos son demasiado débiles para leer bien. Por otra parte, la lista se desborda la página en dos plataformas, y no hay manera de desplazarse hacia arriba.

Una solución es reducir el tamaño del texto. En lugar de usar `NamedSize.Large`, probar uno de los valores más pequeños.

Otra solución parcial se puede encontrar en StackLayout sí mismo: StackLayout define una Espaciado propiedad de tipo doble que indica la cantidad de espacio que dejan entre los niños. Por defecto, es

6.0, pero se puede establecer a algo más pequeño (por ejemplo, cero) para ayudar a garantizar que todos los elementos se ajustan a:

```
content = nuevo StackLayout
{
    Espaciado = 0,
    Los niños =
    {
        nuevo Etiqueta
        {
            text = "Blanco",
            TextColor = Color .Blanco,
            FontSize = fontSize
        },
        ...
    }
}
```

Ahora toda la Etiqueta vistas ocupan sólo la cantidad de espacio vertical como se requiere para el texto. Incluso puede configurar Espaciado a valores negativos para que los elementos se superponen!

Pero la mejor solución se desplaza. El desplazamiento no es compatible automáticamente por StackLayout y debe ser añadido con otro elemento llamado ScrollView, como se verá en la siguiente sección.

Pero hay otro problema con los programas de colores mostrados hasta el momento: ya sea que necesitan para crear explícitamente una variedad de colores y nombres, o explícitamente crear Etiqueta vistas para cada color. Para los programadores, esto es algo tedioso, y por lo tanto un poco desagradable. Podría ser automatizado?

contenido de desplazamiento

Tenga en cuenta que un programa Xamarin.Forms tiene acceso a las bibliotecas de clases base de .NET y se puede utilizar .reflexión NET para obtener información acerca de todas las clases y estructuras definidas en un conjunto, como **Xamarin.Forms.Core**. Esto sugiere que la obtención de los campos estáticos y propiedades de la Color estructura puede ser automatizado.

La mayoría reflexión .NET comienza con una Tipo objeto. Se puede obtener una Tipo objeto para cualquier clase o estructura mediante el uso de la C # tipo de operador. Por ejemplo, la expresión `typeof(color)` devuelve una Tipo objeto para el Color estructura.

En la versión de .NET disponible en el PCL, un método de extensión para la Tipo clase, nombrado `GetTypeInfo`, devuelve una TypeInfo objeto desde el que se puede obtener información adicional. A pesar de que no se requiere en el programa que se muestra a continuación; que necesita otros métodos de extensión definidas para el Tipo clase, nombrado `GetRuntimeFields` y `GetRuntimeProperties`. Estos devuelven los campos y propiedades del tipo en forma de colecciones de `FieldInfo` y `PropertyInfo` objetos. De estos, los nombres, así como los valores de las propiedades se pueden obtener.

Esto se demuestra por la **ReflectedColors** programa. El archivo requiere un `ReflectedColorsPage.cs` utilizando la directiva de `System.Reflection`.

En dos separada para cada declaraciones, el `ReflectedColorsPage` clase recorre todos los campos

y propiedades de la Color estructura. Para todos los miembros estáticos públicos que devuelven Color los valores, los dos bucles llaman CreateColorLabel para crear un Etiqueta con el Color valor y el nombre, y luego añadir que Etiqueta al StackLayout.

Con la inclusión de todos los campos y propiedades estáticas públicas, las listas de programas Color.Transparent, Color.Default, y Color.Accent junto con los 17 campos estáticos que aparecen en el programa anterior. Una separacion CreateColorLabel método crea una Etiqueta ver para cada elemento. Aquí está la lista completa del ReflectedColorsPage clase:

```
clase pública ReflectedColorsPage : Pagina de contenido
{
    público ReflectedColorsPage ()
    {
        StackLayout stackLayout = nuevo StackLayout ();

        // Recorrer los campos de la estructura de color.
        para cada ( FieldInfo info en typeof ( Color ).GetRuntimeFields () )
        {
            // Saltar los colores obsoletos (es decir, mal escritas).
            Si ( Info.GetCustomAttribute < ObsoleteAttribute > (I) != nulo )
                continuar ;

            Si ( Info.IsPublic &&
                  info.IsStatic &&
                  info.FieldType == tipo de ( Color ) )
            {
                stackLayout.Children.Add (
                    CreateColorLabel (( Color ) Info.GetValue ( nulo ), Info.Name));
            }
        }

        // Loop través de las propiedades de la estructura de color.
        para cada ( PropertyInfo info en typeof ( Color ).GetRuntimeProperties () )
        {
            MethodInfo MethodInfo = info.GetMethod;

            Si ( MethodInfo.IsPublic &&
                  MethodInfo.IsStatic &&
                  MethodInfo.ReturnType == tipo de ( Color ) )
            {
                stackLayout.Children.Add (
                    CreateColorLabel (( Color ) Info.GetValue ( nulo ), Info.Name));
            }
        }

        Relleno = nuevo Espesor (5, Dispositivo.OnPlatform (20, 5, 5), 5, 5);

        // Poner el StackLayout en un ScrollView.
        content = nuevo ScrollView
        {
            Content = stackLayout
        };
    }
}
```

```
Etiqueta CreateColorLabel ( Color color, cuerda nombre)
{
    Color backgroundColor = Color .Defecto;

    Si (Color!= Color .Defecto)
    {
        // cálculo de luminancia estándar.
        doble luminancia = 0,30 * color.R +
                           0,59 * + color.G
                           0,11 * color.B;

        backgroundColor = luminancia> 0,5? Color .black: Color .Blanco;
    }

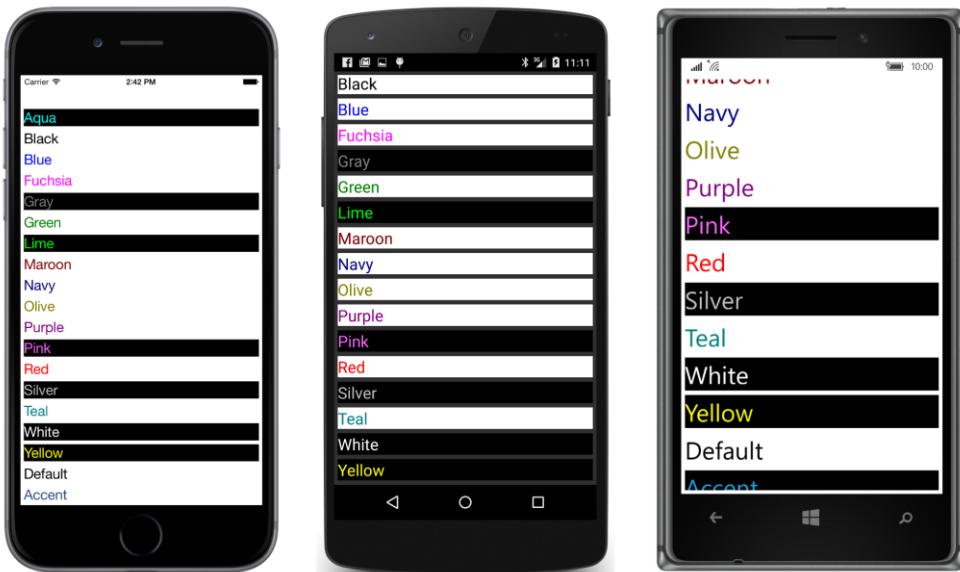
    // crear la etiqueta.
    return new Etiqueta
    {
        Text = nombre,
        TextColor = color,
        Tamaño de Letra = Dispositivo .GetNamedSize ( NamedSize .Grande, tipo de ( Etiqueta )),
        BackgroundColor = backgroundColor
    };
}
```

Hacia el final de la constructor, el StackLayout se establece en el Contenido propiedad de una ScrollView, que se fija luego a la Contenido propiedad de la página.

los CreateColorLabel método en la clase intenta hacer que cada color visible mediante el establecimiento de un fondo de contraste. El método calcula un valor de luminancia en base a una media ponderada estándar de los componentes rojo, verde y azul y se selecciona un fondo de color blanco o negro.

Esta técnica no funcionará para Transparente, por lo que el tema no se puede visualizar en absoluto, y las golosinas método Color.Default como un caso especial y muestra que el color (cuálquiera que sea) en contra de una Color.Default fondo.

Aquí están los resultados, que son todavía muy lejos de ser satisfactoria estéticamente:



Pero se puede efectuar el desplazamiento, debido a que la StackLayout es el hijo de una ScrollView.

StackLayout y ScrollView están relacionados en la jerarquía de clases. StackLayout deriva de Layout, y se recordará que la Disposición <T> clase define la Niños propiedad que Apila-Diseño hereda. el genérico Disposición <T> clase se deriva de la no genérico Diseño clase, y ScrollView También se deriva de esto no genérico Diseño. En teoría, ScrollView es un tipo de objeto, incluso a pesar de que el diseño tiene un solo hijo.

Como se puede ver en la captura de pantalla, el color de fondo de la Etiqueta se extiende a la anchura completa de la StackLayout, que significa que cada Etiqueta es tan ancha como la StackLayout.

Vamos a experimentar un poco para conseguir una mejor comprensión del diseño Xamarin.Forms. Para estos experimentos, es posible que desee dar temporalmente la StackLayout y el ScrollView fondo distintos colores:

```
público ReflectedColorsPage ()
{
    StackLayout stackLayout = nuevo StackLayout
    {
        BackgroundColor = Color.Azul
    };
    ...
    content = nuevo ScrollView
    {
        BackgroundColor = Color.Rojo,
        Content = stackLayout
    };
}
```

objetos de la presentación general tienen fondos transparentes por defecto. A pesar de que ocupan un área de la pantalla, no son directamente visibles. Objetos de diseño dando colores temporales es una gran manera de ver exactamente dónde están en la pantalla. Es una buena técnica de depuración de diseños complejos.

Usted descubrirá que el azul StackLayout asoma en el espacio entre el individuo Etiqueta puntos de vista. Este es el resultado del defecto Espaciado propiedad de StackLayout. Los StackLayout también es visible a través de la Etiqueta para Color.Default, que tiene un fondo transparente.

Intente configurar la HorizontalOptions la propiedad de toda la Etiqueta vistas LayoutOptions.Start:

```
return new Etiqueta
{
    Text = nombre,
    TextColor = color,
    Tamaño de Letra = Dispositivo.GetNamedSize ( NamedSize.Grande, tipo de ( Etiqueta )),
    BackgroundColor = backgroundColor,
    HorizontalOptions = LayoutOptions.Comienzo
};
```

Ahora el fondo azul de la StackLayout es aún más importante porque todo el Etiqueta vistas sólo ocupan tanto espacio horizontal que el texto requiere, y todos ellos son empujados hacia el lado izquierdo. debido a que cada Etiqueta punto de vista es diferente anchura, esta pantalla se ve aún más feo que la primera versión!

Ahora retire el HorizontalOptions el establecimiento de la Etiqueta, y en lugar de establecer una HorizontalOptions sobre el StackLayout:

```
StackLayout stackLayout = nuevo StackLayout
{
    BackgroundColor = Color.Azul,
    HorizontalOptions = LayoutOptions.Comienzo
};
```

Ahora el StackLayout se hace solamente tan ancho como el más amplio etiqueta (al menos en IOS y Android) con el fondo rojo de la ScrollView ahora claramente a la vista.

A medida que comience la construcción de un árbol de objetos visuales, estos objetos adquieren una relación padre-hijo. Un objeto primario se refiere a veces como el *envase* de su niño o los niños porque la ubicación y el tamaño del niño está contenido dentro de su matriz.

Por defecto, HorizontalOptions y VerticalOptions se establecen para LayoutOptions.Fill, lo que significa que cada vista niño intenta llenar el contenedor primario. (Al menos con los contenedores encontrado hasta ahora. Como se verá, otras clases de diseño tienen un comportamiento algo diferente.) Incluso una Etiqueta llena su contenedor principal por defecto, aunque sin un color de fondo, el Etiqueta Parece ser que ocupa sólo el espacio, ya que requiere.

El establecimiento de un punto de vista de HorizontalOptions o VerticalOptions propiedad a LayoutOptions.Start, Centrar, o Fin fuerza efectivamente a la vista para reducir el tamaño hacia abajo, ya sea horizontal, vertical, o ambos a sólo el tamaño de la vista requiere.

UN StackLayout tiene el mismo efecto sobre el tamaño vertical de su hijo: cada niño en una StackLayout ocupa sólo la cantidad de altura que se requiere. Ajuste de la VerticalOptions propiedad de un niño de un Apilar-Diseño a Inicio, Centro, o Fin ¡no tiene efecto! Sin embargo, los puntos de vista del niño todavía se expanden para llenar el ancho de la StackLayout, excepto cuando se les da a los niños una HorizontalOptions otro bien distinto LayoutOptions.Fill.

Si una StackLayout se establece en el Contenido propiedad de una Pagina de contenido, se puede establecer HorizontalOptions o VerticalOptions sobre el StackLayout. Estas propiedades tienen dos efectos: en primer lugar, se encogen el StackLayout anchura o altura (ambos) para el tamaño de sus hijos; y en segundo lugar, que gobiernan en el que el StackLayout está posicionado con respecto a la página.

Si una StackLayout está en un ScrollView, el ScrollView hace que el StackLayout ser sólo tan alto como la suma de las alturas de sus hijos. Así es como el ScrollView puede determinar la forma de desplazarse verticalmente el StackLayout. Puede seguir para establecer el HorizontalOptions propiedad en el Apilar-Diseño para controlar la anchura y la colocación horizontal.

Sin embargo, no debe ajustar VerticalOptions sobre el ScrollView a LayoutOptions-
. Comienzo, Centrar, o Fin. los ScrollView debe ser capaz de desplazarse a su contenido infantil, y la única manera ScrollView puede hacer que está obligando a su hijo (por lo general una StackLayout) para asumir una altura que refleja sólo lo que necesita el niño y después de usar la altura de este niño y su propia altura para calcular cuánto para desplazarse ese contenido. Si se establece VerticalOptions sobre el ScrollView a LayoutOptions.Start, Centro, o Fin, le está diciendo a la eficacia ScrollView ser sólo tan alto como tiene que ser. Pero, ¿qué es esa altura? Porque ScrollView puede desplazarse a su contenido, que no tiene por qué ser cualquier altura determinada, así que en teoría se reducirá a nada. Xamarin.Forms protege frente a esta eventualidad, pero es mejor para usted para evitar código que sugiere algo que no quiere que suceda.

A pesar de poner un StackLayout en un ScrollView es normal, poniendo una ScrollView en un Apilar-Diseño no parece del todo bien. En teoría, el StackLayout obligará a la ScrollView a tener una altura de sólo lo que requiere, y que la altura requerida es básicamente cero. Una vez más, Xamarin.Forms protege frente a esta eventualidad, pero se debe evitar este tipo de código.

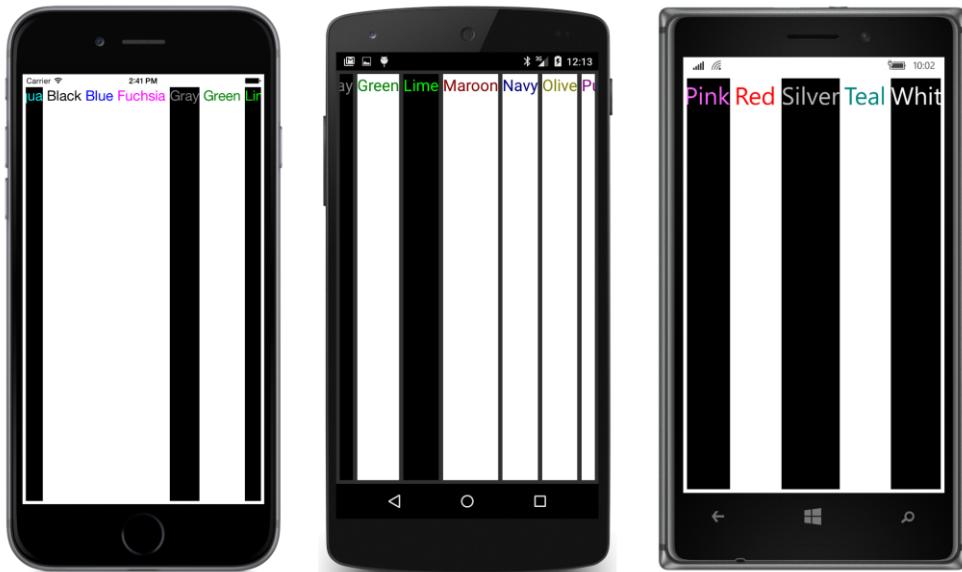
Hay una forma correcta de poner un ScrollView en un StackLayout es decir, en total acuerdo con los principios de diseño Xamarin.Forms, y que se demostrará en breve.

La discusión anterior se aplica a orientado verticalmente StackLayout y ScrollView elementos. StackLayout tiene una propiedad denominada Orientación que se puede establecer a un miembro de la StackOrientation enumeración- Vertical (el valor por defecto) o Horizontal. Similar, ScrollView También tiene una Orientación propiedad que define a un miembro de la ScrollOrientation enumeración. Prueba esto:

```
público ReflectedColorsPage ()  
{  
    StackLayout stackLayout = nuevo StackLayout  
    {  
        Orientación = StackOrientation.Horizontal  
    };  
    ...  
    content = nuevo ScrollView
```

```
{  
    Orientación = ScrollOrientation.Horizontal,  
    Content = stackLayout  
};  
}
```

Ahora el Etiqueta vistas están apilados horizontalmente, y el ScrollView llena la página verticalmente pero permite el desplazamiento horizontal de la StackLayout, que verticalmente llena el ScrollView:



Se ve bastante raro con las opciones de diseño verticales predeterminadas, pero los que podría fijarse para que se vea un poco mejor.

La opción expande

Usted probablemente ha notado que la `HorizontalOptions` y `VerticalOptions` propiedades son plurales, como si hay más de una opción. Estas propiedades se establecen generalmente a un campo estático de la `LayoutOptions` estructura-otro plural.

Las discusiones se han centrado hasta ahora en los siguientes estática de sólo lectura `LayoutOptions` campos que devuelven valores predefinidos de `LayoutOptions`:

- `LayoutOptions.Start`
- `LayoutOptions.Center`
- `LayoutOptions.End`

- LayoutOptions.Fill

El valor predeterminado establecido por el Ver clase-es LayoutOptions.Fill, lo que significa que la vista llena su contenedor.

Como hemos visto, una VerticalOptions establecer en una Etiqueta no hace una diferencia cuando el Etiqueta es un hijo de la vertical StackLayout. Los StackLayout sí limita la altura de sus hijos sólo a la altura que requieren, para que el niño no tiene libertad para moverse verticalmente dentro de esa ranura.

Esté preparado para esta regla que modificar ligeramente!

Los LayoutOptions estructura cuenta con cuatro campos de sólo lectura estáticas adicionales que no se discuten sin embargo:

- LayoutOptions.StartAndExpand
- LayoutOptions.CenterAndExpand
- LayoutOptions.EndAndExpand
- LayoutOptions.FillAndExpand

LayoutOptions también define dos propiedades de la instancia, nombrados Alineación y Expande. Los cuatro casos de LayoutOptions devuelto por los campos estáticos que terminan con AndExpand todos tienen la Ex-
marcas de fáctica propiedad establecida en cierto.

Esta expande propiedad está reconocido sólo por StackLayout. Puede ser muy útil para la gestión del diseño de la página, pero puede ser confuso en el primer encuentro. Estos son los requisitos para expande a desempeñar un papel en la vertical StackLayout:

- El contenido de la StackLayout deberá tener una altura total que es menor que la altura de la StackLayout sí mismo. En otras palabras, un poco de espacio vertical extra no utilizado debe existir en el Apilar-Diseño.
- Ese primer requisito implica que la vertical, StackLayout No puede tener su propio Vertical-opciones propiedad establecida en Inicio, Centro, o Fin porque eso sería hacer que el StackLayout tener una altura igual a la altura total de sus hijos, y que no tendría ningún espacio adicional.
- Al menos un hijo de la StackLayout debe tener una VerticalOptions con el establecimiento de la expande propiedad establecida en cierto.

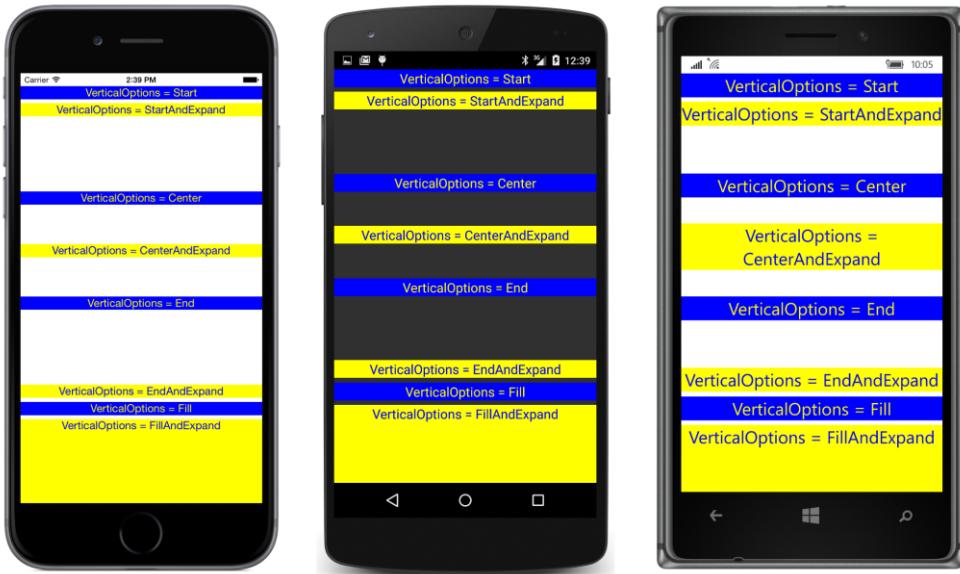
Si se cumplen estas condiciones, el StackLayout asigna el espacio vertical extra por igual entre todos los niños que tienen una VerticalOptions ajuste con expande igual a cierto. Cada uno de estos niños para crear una ranura más grande en el StackLayout de lo normal. Cómo el niño ocupa ese espacio depende de la Alineación ajuste de la LayoutOptions valor: Inicio, Centro, Fin, o Llenar.

He aquí un programa, llamado **VerticalOptionsDemo**, que utiliza la reflexión para crear Etiqueta objetos con toda la posible VerticalOptions ajustes en la vertical StackLayout. Los colores de fondo y primer plano se alternan de manera que pueda ver exactamente cuánto espacio cada Etiqueta ocupa. los

programa utiliza Language Integrated Query (LINQ) para ordenar los campos de la LayoutOptions estructura de una manera visualmente más esclarecedor:

```
p&gt;<code>p&gt;ublico clase VerticalOptionsDemoPage : Pagina de contenido<code>{<code>p&gt;ublico VerticalOptionsDemoPage ()<code>{<code>p&gt; Color [] = {Colores Color .Amarillo, Color .Blue};<code>En t flipFlopper = 0;<code> // crear etiquetas ordenados por la propiedad LayoutAlignment.<code>IEnumerable < Etiqueta > etiquetas =<code> de campo en typeof ( LayoutOptions ) .GetRuntimeFields ()<code>dnde field.IsPublic && field.IsStatic<code> OrdenarPor (( LayoutOptions ) Field.GetValue ( nulo )) .Alineaci&on<code>seleccione nuevo Etiqueta<code>{<code>text = "VerticalOptions =" + Field.Name,<code>VerticalOptions = ( LayoutOptions ) Field.GetValue ( nulo ),<code>HorizontalTextAlignment = Alineaci&on del texto .Centrar,<code>Tama&o de Letra = Dispositivo .GetNamedSize ( NamedSize .Medio, tipo de ( Etiqueta )),<code>TextColor = colores [flipFlopper],<code>BackgroundColor = colores [flipFlopper = 1 - flipFlopper]<code>};<code> // Traslado al StackLayout.<code>StackLayout stackLayout = nuevo StackLayout ();<code>para cada ( Etiqueta etiqueta en etiquetas )<code>{<code>stackLayout.Children.Add (etiqueta);<code>}<code>Relleno = nuevo Espesor (0, Dispositivo .OnPlatform (20, 0, 0), 0, 0);<code>Content = stackLayout;<code>}</pre>
```

Es posible que desee estudiar los resultados un poco:



Los Etiqueta puntos de vista con texto amarillo en fondos azules son los que tienen `VerticalOptions` propiedades ajustado a `LayoutOptions` valores *sin el expande set bandera*. Si el expande bandera no está definida en el

`LayoutOptions` valor de un elemento en una línea vertical `StackLayout`, el `VerticalOptions` ajuste se ignora. Como se puede ver, el Etiqueta ocupa sólo la cantidad de espacio vertical, ya que necesita en la vertical `StackLayout`.

La altura total de los niños en este `StackLayout` es menor que la altura de la `StackLayout`, entonces el `StackLayout` tiene espacio adicional. Contiene cuatro hijos con su `VerticalOptions` propiedades ajustado a `LayoutOptions` valores con el expande set bandera, por lo que este espacio adicional se asigna por igual entre los cuatro hijos.

En estos cuatro casos, la Etiqueta puntos de vista con el texto azul sobre fondo de color amarillo-la Alineación propiedad de la `LayoutOptions` valor indica cómo el niño se alinea dentro de la zona que incluye el espacio extra. La primera de ellas, con el `VerticalOptions` propiedad establecida en `LayoutOptions.StartAnd-`

Expandir -es por encima de este espacio adicional. El segundo (`CenterAndExpand`) está en el centro del espacio extra. El tercero (`EndAndExpand`) está por debajo del espacio extra. Sin embargo, en todos estos tres casos, la Etiqueta está recibiendo sólo la cantidad de espacio vertical, ya que necesita, como se indica por el color de fondo. El resto del espacio pertenece a la `StackLayout`, que muestra el color de fondo de la página.

El último Etiqueta tiene su `VerticalOptions` propiedad establecida en `LayoutOptions.FillAndExpand`. En este caso, el Etiqueta ocupa toda la ranura incluyendo el espacio adicional, ya que la gran superficie de fondo amarillo indica. El texto está en la parte superior de esta zona; eso se debe a la configuración por defecto de `Vertical-`

Alineación del texto es `TextAlignment.Start`. Configurarlo para algo más para colocar el texto verticalmente dentro de la zona.

los expande propiedad de LayoutOptions juega un papel sólo cuando la vista es un niño de un Apilar-Diseño. En otros contextos, se ignora.

Marco y BoxView

Dos simples vistas rectangulares son a menudo útiles para los propósitos de la presentación:

los BoxView es un rectángulo lleno. Se deriva de Ver y define una Color propiedad con un valor predeterminado de Color.Default eso es transparente por defecto.

los Marco Muestra un borde rectangular que rodea parte del contenido. Marco deriva de Diseño por medio de ContentView, del que hereda una Contenido propiedad. El contenido de una Marco puede ser una única vista o una disposición que contiene un montón de puntos de vista. De VisualElement, Frame hereda una Espalda-groundColor propiedad que es blanco en el iPhone, pero transparente en Android y Windows Phone. De Diseño, Marco hereda una Relleno propiedad que se inicializa a 20 unidades en todos los lados para dar el contenido un poco de espacio para respirar. Marco en sí define una HasShadow propiedad que es cierto por defecto (pero la sombra se muestra sólo en iOS dispositivos) y una OutlineColor propiedad que es transparente por defecto, pero no afecta a la sombra de iOS, que siempre es negro y siempre visible cuando HasShadow se establece en cierto.

Ambos Marco delinear y la BoxView es transparente por defecto, por lo que podría ser un poco incierto cómo colorearlos sin recurrir a diferentes colores para diferentes plataformas. Una buena opción es Color.Accent, que está garantizado a aparecer independientemente. O bien, puede tomar el control de la coloración del fondo, así como la Marco esquema y BoxView.

Si el BoxView o Marco No se ve limitada en el tamaño de cualquier manera, es decir, si no es en una StackLayout y tiene su HorizontalOptions y VerticalOptions conjunto de valores de defecto LayoutOptions-. Llenar vistas -estos expanden para llenar sus contenedores.

Por ejemplo, aquí hay un programa que ha centrado una Etiqueta establecido en el Contenido propiedad de una Marco:

```
público clase FramedTextPage : Pagina de contenido
{
    pública FramedTextPage ()
    {
        Relleno = nuevo Espesor (20);
        content = nuevo Marco
        {
            OutlineColor = Color .Acento,
            content = nuevo Etiqueta
            {
                text = "Me han tendido una trampa",
                Tamaño de Letra = Dispositivo .GetNamedSize ( NamedSize .Grande, tipo de ( Etiqueta )),
                HorizontalOptions = LayoutOptions .Centrar,
                VerticalOptions = LayoutOptions .Centrar
            }
        };
    }
}
```

```
    }  
}
```

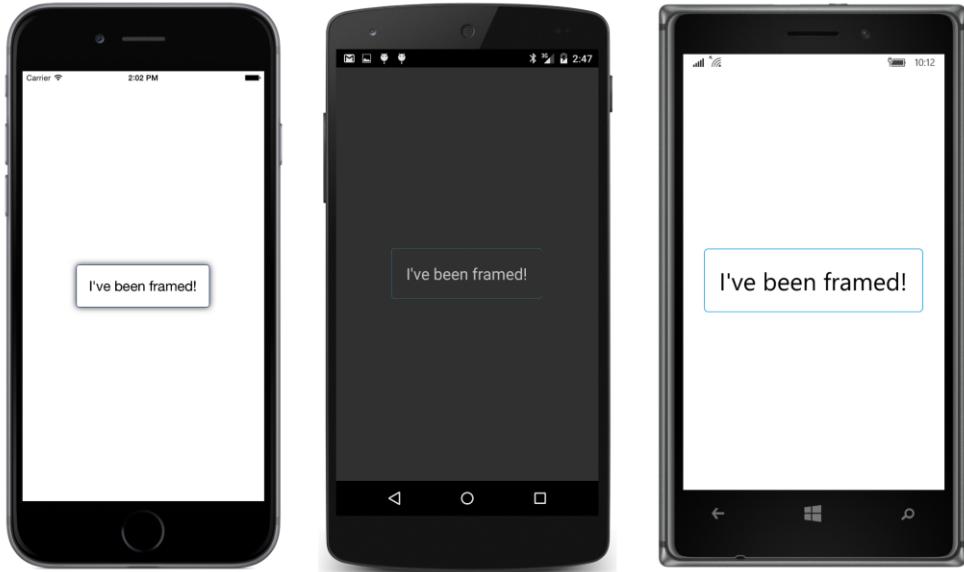
los Etiqueta está centrado en el Marco, pero el Marco ocupe toda la página, y que ni siquiera podría ser capaz de ver el Marco claramente si la página no se había dado una Relleno de 20 en todos los lados:



Para mostrar texto enmarcado centrada, que desea establecer la `HorizontalOptions` y `VerticalOptions` propiedades en el Cuadro (en lugar de la Etiqueta) a `LayoutOptions.Center`:

```
clase pública FramedTextPage : Pagina de contenido  
{  
    público FramedTextPage ()  
    {  
        Relleno = nuevo Espesor (20);  
        content = nuevo Marco  
        {  
            OutlineColor = Color .Acento,  
            HorizontalOptions = LayoutOptions .Centrar,  
            VerticalOptions = LayoutOptions .Centrar,  
            content = nuevo Etiqueta  
            {  
                text = "Me han tendido una trampa!",  
                Tamaño de Letra = Dispositivo .GetNamedSize ( NamedSize .Grande, tipo de ( Etiqueta ))  
            }  
        };  
    }  
}
```

Ahora el Marco abraza el texto (pero con relleno por defecto de 20 unidades de bastidor) en el centro de la página:



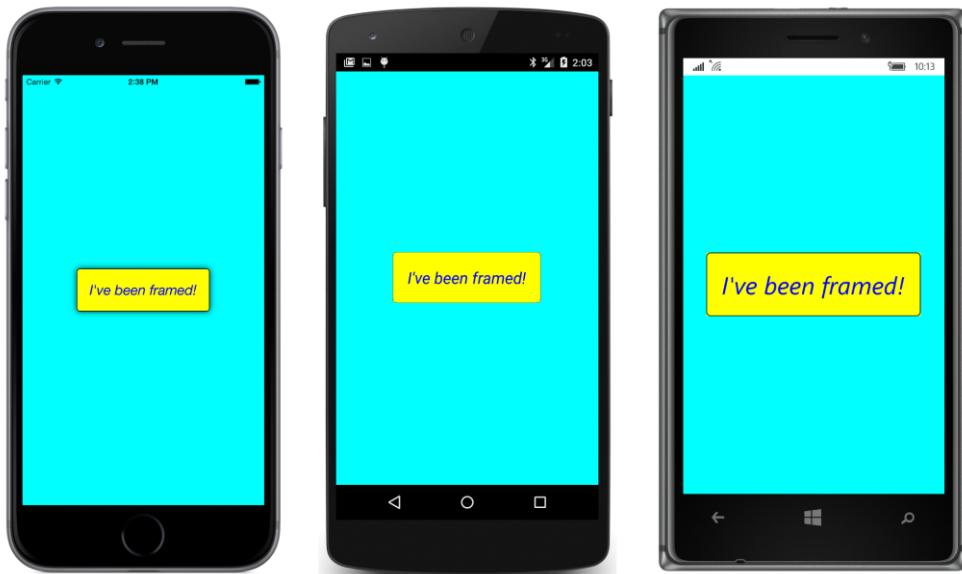
La versión de **FramedText** incluido con el código de ejemplo para este capítulo uso de la libertad de dar todo un color personalizado:

```
público clase FramedTextPage : Página de contenido
{
    público FramedTextPage ()
    {
        BackgroundColor = Color .Aqua;

        content = nuevo Marco
        {
            OutlineColor = Color .Negro,
            BackgroundColor = Color .Amarillo,
            HorizontalOptions = LayoutOptions .Centrar,
            VerticalOptions = LayoutOptions .Centrar,

            content = nuevo Etiqueta
            {
                text = "Me han tendido una trampa",
                Tamaño de Letra = Dispositivo .GetNamedSize ( NamedSize .Grande, tipo de ( Etiqueta )),
                FontAttributes = FontAttributes .Itálico,
                TextColor = Color .Azul
            }
        };
    }
}
```

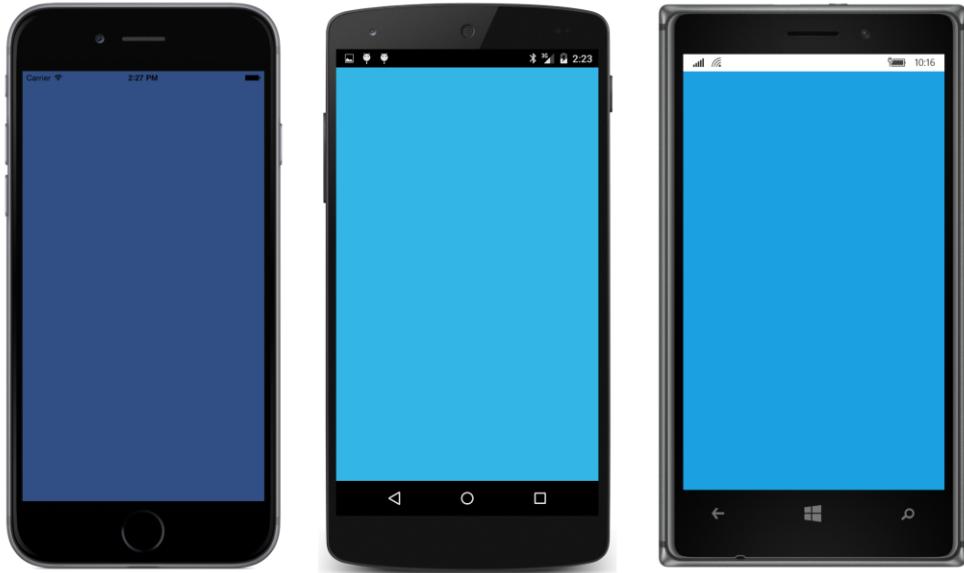
El resultado se ve más o menos el mismo en las tres plataformas:



Pruebe a establecer una BoxView al Contenido propiedad de una Pagina de contenido, al igual que:

```
clase pública SizedBoxViewPage : Página de contenido
{
    público SizedBoxViewPage ()
    {
        content = nuevo BoxView
        {
            color = Color .Acento
        }
    }
}
```

Asegúrese de ajustar el Color propiedad para que pueda verlo. los BoxView llena toda el área de su recipiente, tal como Etiqueta lo hace con su valor por defecto HorizontalOptions o VerticalOptions ajustes:

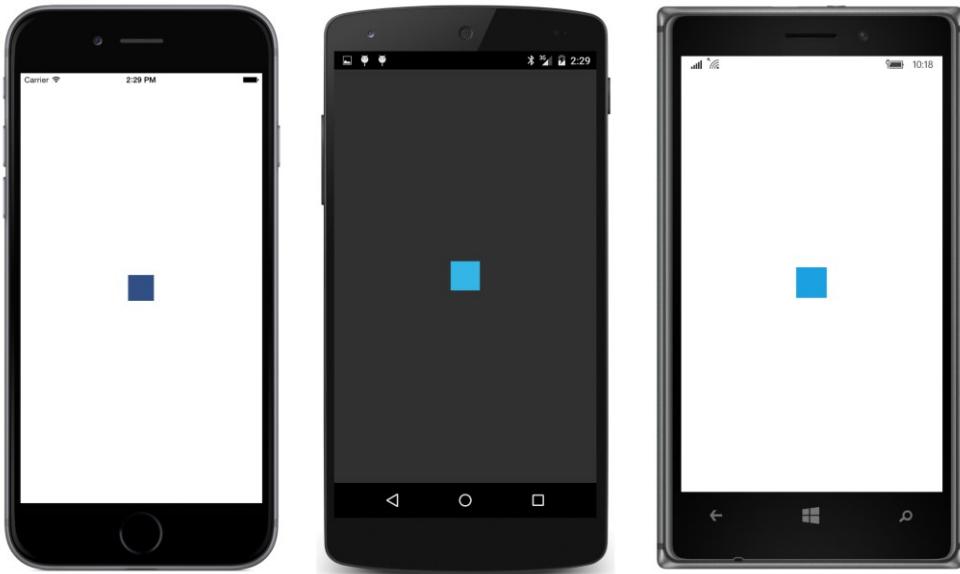


Es incluso que subyace en la barra de estado iOS!

Ahora intente configurar el `HorizontalOptions` y `VerticalOptions` propiedades de la `BoxView` a algo distinto Llenar, como en este ejemplo de código:

```
público clase SizedBoxViewPage : Página de contenido
{
    público SizedBoxViewPage ()
    {
        content = nuevo BoxView
        {
            color = Color .Acento,
            HorizontalOptions = LayoutOptions .Centrar,
            VerticalOptions = LayoutOptions .Centrar
        };
    }
}
```

En este caso, el `BoxView` asumirá sus dimensiones por defecto de 40 unidades cuadrado:



los BoxView es ahora de 40 unidades cuadradas debido a que la BoxView inicializa su WidthRequest y HeightRequest propiedades a 40. Estas dos propiedades requieren una pequeña explicación:

VisualElement define Anchura y Altura propiedades, pero estas propiedades son de sólo lectura. **VisualElement** también define WidthRequest y HeightRequest propiedades que son tanto ajustable y gettable. Normalmente, todas estas propiedades se inicializan a -1 (lo que significa eficazmente que no están definidos), pero algunos Ver derivados, tales como **BoxView**, selecciona el WidthRequest y HeightRequest propiedades a valores específicos.

Después de una página ha organizado el trazado de sus hijos y rendido todos los elementos visuales, las Anchura y Altura propiedades indican dimensiones reales de cada vista-área que la vista ocupa en la pantalla. Porque Anchura y Altura son de sólo lectura, que son sólo para fines informativos. (Capítulo 5, "Tratar con los tamaños", describe cómo trabajar con estos valores.)

Si quieres una vista que es un tamaño específico, se puede establecer el WidthRequest y HeightRequest propiedades. Sin embargo, estas propiedades indican (como su nombre indica) una *pedido* tamaño o una *privilegiado* tamaño. Si se permite que la vista para llenar su recipiente, estas propiedades serán ignorados.

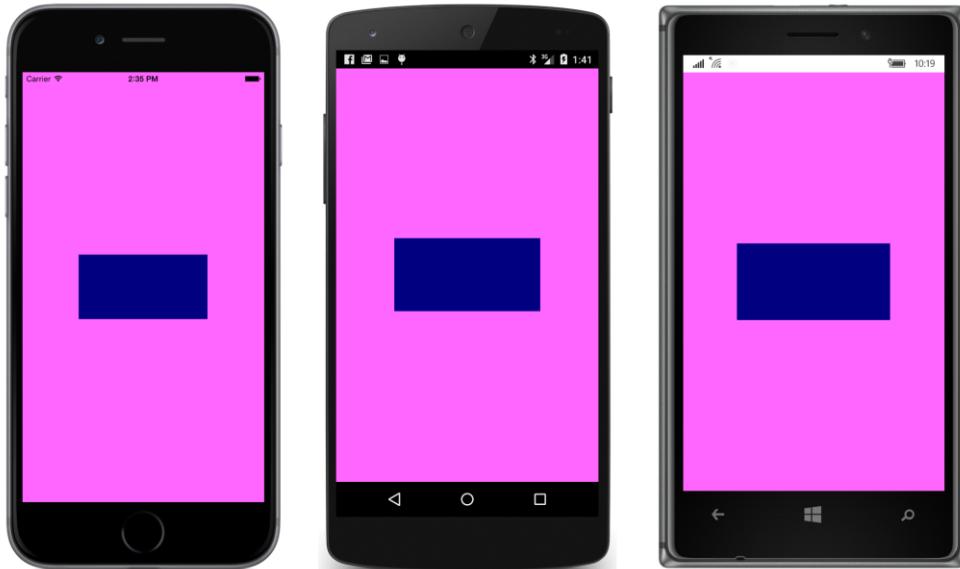
BoxView establece su tamaño predeterminado de valores de 40 reemplazando el OnSizeRequest método. Se puede pensar en estos ajustes como un tamaño que BoxView le gustaría ser si nadie más tiene alguna opinión en la materia. Ya hemos visto que WidthRequest y HeightRequest se ignoran cuando el BoxView se le permite llenar la página. Los WidthRequest entra en acción si el HorizontalOptions se establece en LayoutOptions.Left, Center, o Derecha, o si el BoxView es un hijo de un eje horizontal StackLayout. Los HeightRequest se comporta de manera similar.

Aquí está la versión de la **SizedBoxView** programa incluido con el código de este capítulo:

```
público clase SizedBoxViewPage : Pagina de contenido
{
    público SizedBoxViewPage ()
    {
        BackgroundColor = Color.Rosado;

        content = nuevo BoxView
        {
            color = Color.Armada,
            HorizontalOptions = LayoutOptions.Centralizar,
            VerticalOptions = LayoutOptions.Centralizar,
            WidthRequest = 200,
            HeightRequest = 100
        };
    }
}
```

Ahora tenemos una BoxView con ese tamaño específico y los colores establecidos de forma explícita:



Vamos a usar tanto Marco y BoxView en una lista de color mejorada. los **Colorblocks** programa tiene un constructor de página que es prácticamente idéntica a la de **ReflectedColors**, excepto que llama a un método llamado CreateColorView más bien que CreateColorLabel. He aquí que el método:

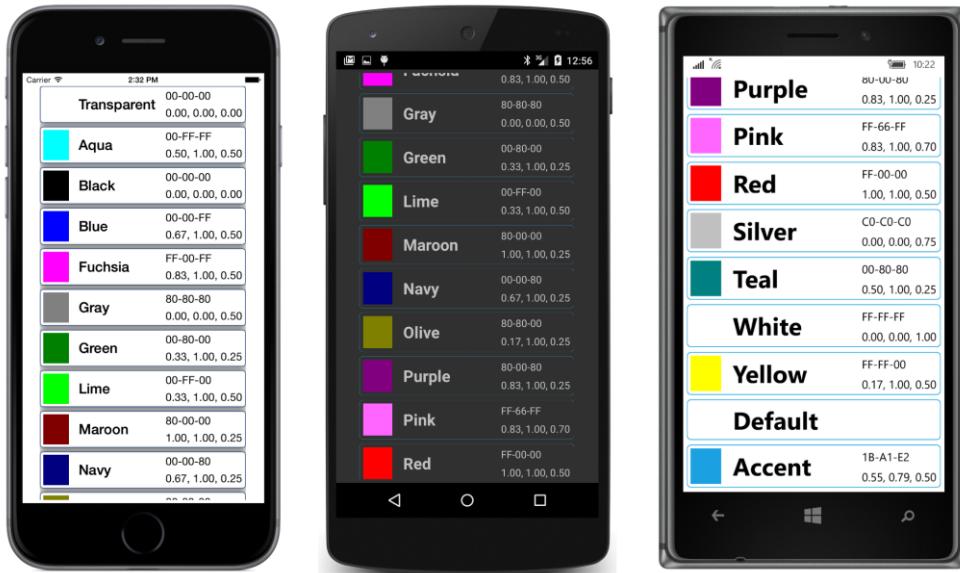
```
clase ColorBlocksPage : Pagina de contenido
{
    ...
}

Ver CreateColorView ( Color color, cuerda nombre)
{
```

```
return new Marco
{
    OutlineColor = Color .Acento,
    Relleno = nuevo Espesor (5),
    content = nuevo StackLayout
    {
        Orientación = StackOrientation .Horizontal,
        Espaciado = 15,
        Los niños =
        {
            nuevo BoxView
            {
                Color = Color
            },
            nuevo Etiqueta
            {
                Text = nombre,
                Tamaño de Letra = Dispositivo .GetNamedSize ( NamedSize .Grande, tipo de ( Etiquetas )),
                FontAttributes = FontAttributes .Negrita,
                VerticalOptions = LayoutOptions .Centrar,
                HorizontalOptions = LayoutOptions .StartAndExpand
            },
            nuevo StackLayout
            {
                Los niños =
                {
                    nuevo Etiqueta
                    {
                        text = Cuerda .Formato( "{0: X2} - {1: X2} - {2: X2}" ,
                            ( Ent ) ( 255 * color.R ),
                            ( Ent ) ( 255 * color.G ),
                            ( Ent ) ( 255 * color.B )),
                        VerticalOptions = LayoutOptions .CenterAndExpand,
                        IsVisible = color! = Color .Defecto
                    },
                    nuevo Etiqueta
                    {
                        text = Cuerda .Formato( "{0: F2}, {1: F2}, {2: F2}" ,
                            color.Hue,
                            saturación de color,
                            color.Luminosity),
                        VerticalOptions = LayoutOptions .CenterAndExpand,
                        IsVisible = color! = Color .Defecto
                    }
                },
                HorizontalOptions = LayoutOptions .Fin
            }
        }
    };
}
```

los `CreateColorView` método devuelve una `Marco` que contiene una horizontal `StackLayout` con un `Caja`-
Ver que indica el color, una Etiqueta para el nombre del color, y otro `StackLayout` con dos más Etiqueta vistas para la composición y
el RGB La saturación de color, y Luminosidad valores. Las pantallas RGB y HSL tienen sentido para el `Color.Default` valor, de modo
que interna `StackLayout` tiene su `Es visible` propiedad establecida en falso en ese caso. Los `StackLayout` todavía existe, pero se
ignora cuando se procesa la página.

El programa no sabe qué elemento va a determinar la altura de cada color del artículo -el Caja-
Ver, el Etiqueta con el nombre del color, o los dos Etiqueta puntos de vista con los valores RGB y HSL- por lo que todos los centros Etiqueta puntos de
vista. Como se puede ver, el `BoxView` se expande en altura para acomodar la altura del texto:



Ahora bien, esta es una lista de colores desplazable que está empezando a haber algo que podamos tomar un poco de orgullo en.

Un ScrollView en un StackLayout?

Es común para poner una `StackLayout` en un `ScrollView`, pero se puede poner una `ScrollView` en un `StackLayout`? Y ¿por qué vas a querer?

Es una regla general en los sistemas de diseño como el de Xamarin.Forms que no se puede poner un desplazamiento en una pila.
UN `ScrollView` debe tener una altura específica para calcular la diferencia entre la altura de su contenido y de su propia altura. Esta
diferencia es la cantidad que la `ScrollView` puede desplazarse a su contenido. Si el `ScrollView` está en un `StackLayout`, no consigue que la altura
específica. los `StackLayout`

quiero que el ScrollView ser lo más corto posible, y eso es ya sea la altura de la ScrollView contenidos o cero, y tampoco funciona la solución.

Así que ¿por qué quiere una ScrollView en un StackLayout ¿de todas formas?

A veces es exactamente lo que necesita. Considere un lector de libros electrónicos primitiva que implementa el desplazamiento. Es posible que desee una Etiqueta en la parte superior de la página mostrando siempre el título del libro, seguido de una ScrollView que contiene una StackLayout con el contenido del libro en sí. Sería conveniente para que Etiqueta y el ScrollView ser hijos de una StackLayout que llena la página.

Con Xamarin.Forms, tal cosa es posible. Si se le da la ScrollView un VerticalOptions ajuste de LayoutOptions.FillAndExpand, que de hecho puede ser un hijo de una StackLayout. Los StackLayout fuera dará a la ScrollView todo el espacio extra no es requerido por los otros niños, y la ScrollView entonces tendrá una altura específica. Curiosamente, Xamarin.Forms protege contra otras configuraciones de esa VerticalOptions propiedad, por lo que funciona con lo que se establece a.

Los **Gato negro** proyecto muestra el texto del cuento de Edgar Allan Poe "El Gato Negro", que se almacena en un archivo de texto denominado TheBlackCat.txt en un formato de una línea-párrafo.

Cómo hace el **Gato negro** programa de acceso al archivo con este cuento? Tal vez el método más sencillo es la de insertar el archivo de texto a la derecha en el programa ejecutable o en el caso de una aplicación correcta Xamarin.Forms en el portátil Biblioteca de DLL clase. Estos archivos se conocen como *recursos incrustados*, y eso es lo TheBlackCat.txt archivo está en este programa.

Para hacer un recurso incrustado en Visual Studio o Xamarin de estudio, es probable que primero desea crear una carpeta en el proyecto seleccionando el **Agregar > Nueva carpeta** opción en el menú proyecto. Una carpeta de archivos de texto que se podría llamar **Textos**, por ejemplo. La carpeta es opcional, pero ayuda a organizar los activos del programa. Entonces, en esa carpeta, puede seleccionar **Añadir > elemento existente** en Visual Studio o **Añadir > Añadir archivos** en Xamarin Studio. Navegue hasta el archivo, selecciónelo y haga clic **Añadir** en Visual Studio o **Abierto** en Xamarin Studio.

Ahora aquí está la parte importante: Una vez que el archivo es parte del proyecto, abra el **propiedades** diálogo desde el menú asociado con el archivo. Especificar que el **construir Acción** para el archivo es **EmbeddedResource**. Este es un paso fácil de olvidar, pero es esencial.

Esto se hizo para el **Gato negro** proyecto, y en consecuencia el archivo TheBlackCat.txt se incrusta en el archivo BlackCat.dll.

En el código, el archivo puede ser recuperada mediante una llamada al GetManifestResourceStream método definido por la Montaje clase en el System.Reflection espacio de nombres. Para obtener el montaje del PCL, todo lo que necesita hacer es obtener la Tipo de cualquier clase definida en el conjunto. Puedes usar tipo de con el tipo de página que ha derivado de Pagina de contenido o GetType en la instancia de esa clase. Luego llame GetType- información en este Tipo objeto. Montaje es una propiedad de la resultante TypeInfo objeto:

```
Montaje montaje = GetType () GetTypeinfo () Asamblea.;
```

En el GetManifestResourceStream método de Montaje, tendrá que especificar el nombre de la

recurso. Para recursos incrustados, que el nombre no es el nombre del archivo del recurso, pero el *Identificación de recursos*.

Es fácil confundir estos, ya que la ID puede tener un aspecto vagamente a un nombre de archivo completo.

El ID de recurso comienza con el nombre por defecto de la asamblea. Este no es el espacio de nombres de .NET! Para obtener el nombre por defecto de la asamblea en Visual Studio, seleccione **propiedades** En el menú proyecto, y en el diálogo de propiedades, seleccione **Biblioteca** a la izquierda y buscar la **espacio de nombres predeterminado**

campo. En Xamarin Studio, seleccione **opciones** En el menú proyecto, y en el **Opciones del proyecto** diálogo, seleccione **Ajustes principales** a la izquierda, y buscar un campo etiquetado **Por defecto Espacio de nombres**.

Para el **Gato negro** proyecto, ese espacio de nombres por defecto es la misma que la asamblea: "BlackCat". Sin embargo, en realidad se puede establecer que el espacio de nombres predeterminado a lo que quieras.

El ID de recurso comienza con ese espacio de nombres por defecto, seguido de un punto, seguido por el nombre de la carpeta que podría haber utilizado, seguido por otro período y el nombre del archivo. Para este ejemplo, el ID de recurso es "BlackCat.Texts.TheBlackCat.txt", y eso es lo que va a pasar a la **GetManifestResourceStream** método en el código. El método devuelve un .NET Corriente objeto, y de que una **StreamReader** se pueden crear para leer las líneas de texto.

Es una buena idea usar declaraciones con el Corriente objeto devuelto desde **GetManifestResourceStream** y el **StreamReader** objetar porque eso va a disponer adecuadamente de los objetos cuando están ya no son necesarios o si plantean excepciones.

Para fines de diseño, la **BlackCatPage** constructor crea dos **StackLayout** objetos: **mainStack** y **textStack**. La primera línea del archivo (que contiene título y el autor de la historia) se convierte en una negrita y centrado Etiqueta en **mainStack**; todas las líneas subsiguientes van en **textStack**. Los **mainStack** instancia contiene también una **ScrollView** con **textStack**.

```
clase BlackCatPage : Pagina de contenido
{
    público BlackCatPage ()
    {
        StackLayout mainStack = nuevo StackLayout ();
        StackLayout textStack = nuevo StackLayout ();
        {
            Relleno = nuevo Espesor (5),
            Espaciado = 10
        };

        // Obtener acceso al recurso de texto.
        Montaje montaje = GetType () GetTypeInfo () Asamblea. ;
        cuerda = recursos "BlackCat.Texts.TheBlackCat.txt" ;

        utilizando ( Corriente flujo = assembly.GetManifestResourceStream (recurso))
        {
            utilizando ( StreamReader lector = nuevo StreamReader (corriente))
            {
                bool gotTitle = falso ;
                cuerda linea;

                // Leer en una línea (que en realidad es un párrafo).
            }
        }
    }
}
```

```

mientras ( nulo != (Línea = reader.ReadLine ()))
{
    Etiqueta etiqueta = nuevo Etiqueta
    {
        Texto = linea,
        // texto Negro de libros electrónicos!
        TextColor = Color .Negro
    };

    Si ( ! GotTitle)
    {
        // Añadir primera etiqueta (el título) a mainStack.
        label.HorizontalOptions = LayoutOptions .Centrar;
        label.FontSize = Dispositivo .GetNamedSize ( NamedSize .Medium, etiqueta);
        label.FontAttributes = FontAttributes .Negrita;
        mainStack.Children.Add (etiqueta);
        gotTitle = cierto ;
    }
    más
    {
        // Añadir etiquetas posteriores a textStack.
        textStack.Children.Add (etiqueta);
    }
}

// Poner el textStack en un ScrollView con FillAndExpand.
ScrollView ScrollView = nuevo ScrollView
{
    Content = textStack,
    VerticalOptions = LayoutOptions .FillAndExpand,
    Relleno = nuevo Espesor (5, 0),
};

// Añadir el ScrollView como un segundo niño de mainStack.
mainStack.Children.Add (ScrollView);

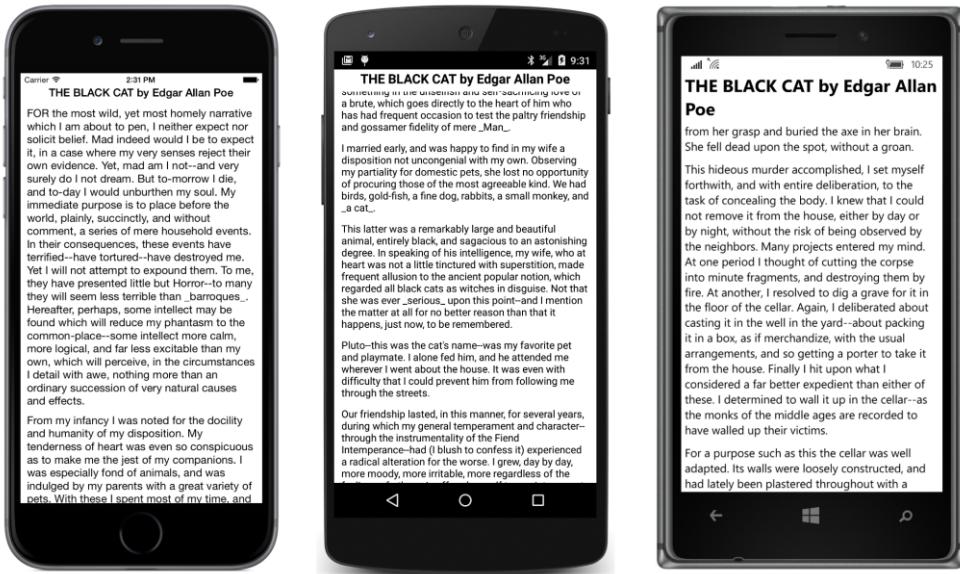
// Establecer contenido de la página a mainStack.
Content = mainStack;

// Fondo blanco para los libros electrónicos!
BackgroundColor = Color .Blanco;

// Añadir un poco de relleno IOS para la página.
Relleno = nuevo Espesor (0, Dispositivo .OnPlatform (20, 0, 0), 0, 0);
}
}

```

Debido a que este es básicamente un lector de libros electrónicos, y los seres humanos han estado leyendo el texto negro sobre papel blanco durante cientos de años, las Color de fondo La página se establece en blanco y la Color de texto de cada Etiqueta se establece en negro:



Gato negro es una aplicación de PCL. También es posible escribir este programa mediante un proyecto activo compartido en lugar de un PCL. Para demostrarlo, una **BlackCatSap** proyecto se incluye con el código de este capítulo. Sin embargo, debido a que el recurso de hecho se convierte en parte del proyecto de aplicación, necesitará el espacio de nombres predeterminado para la aplicación, y eso es diferente para cada plataforma. El código para establecer la variable de recursos se ve así:

```
# Si _IOS_
    cuerda = recursos "BlackCatSap.iOS.Texts.TheBlackCat.txt";
# elif _ANDROIDE_
    cuerda = recursos "BlackCatSap.Droid.Texts.TheBlackCat.txt";
# elif WINDOWS_UWP
    cuerda = recursos "BlackCatSap.UWP.Texts.TheBlackCat.txt";
# elif WINDOWS_APP
    cuerda = recursos "BlackCatSap.Windows.Texts.TheBlackCat.txt";
# elif WINDOWS_PHONE_APP
    cuerda = recursos "BlackCatSap.WinPhone.Texts.TheBlackCat.txt";
# terminara si
```

Si usted está teniendo problemas hace referencia a un recurso incrustado, puede que esté utilizando un nombre incorrecto. Trate de llamar `GetManifestResourceNames` sobre el `Montaje` objeto para obtener una lista de los ID de los recursos de todos los recursos incrustados.

Capítulo 5

Tratar con tamaños

Ya has visto algunas referencias a tamaños en relación con diversos elementos visuales:

- La barra de estado iOS tiene una altura de 20, que se puede ajustar para una Relleno establecer en la página.
- los BoxView establece su anchura predeterminada y la altura a 40.
- El valor por defecto Relleno Dentro de un Marco es 20.
- El valor por defecto Espaciado propiedad en el StackLayout es 6.

Y luego está Device.GetNamedSize, que para varios miembros de la NamedSize enumeración devuelve un número apropiado dependiente de la plataforma para Tamaño de fuente valores para una Etiqueta o Botón.

¿Cuáles son esos números? ¿Cuáles son sus unidades? ¿Y cómo inteligentemente establecer las propiedades que requieren tamaños a otros valores?

Buena pregunta. Como hemos visto, las diferentes plataformas tienen diferentes tamaños de pantalla y diferentes tamaños de texto, y todos muestran una cantidad diferente de texto en la pantalla. Es la cantidad de texto que algo una aplicación Xamarin.Forms puede anticipar o control? E incluso si es posible, se trata de una práctica de programación correcta? En caso de que una aplicación ajustar el tamaño de las fuentes para lograr una densidad deseada en la pantalla?

En general, cuando se programa una aplicación Xamarin.Forms, es mejor no acercarse demasiado a las dimensiones numéricas reales de los objetos visuales. Es preferible confiar Xamarin.Forms y las plataformas individuales para tomar las mejores opciones por defecto.

Sin embargo, hay veces en que un programador necesita saber algo sobre el tamaño de los objetos visuales particulares y el tamaño de la pantalla en la que aparecen.

Píxeles, puntos, dps, salsas, y UDE

pantallas de vídeo consisten en una matriz rectangular de píxeles. Cualquier objeto que se muestra en la pantalla también tiene un tamaño de pixel. En los primeros días de las computadoras personales, los programadores el tamaño y posición de objetos visuales en unidades de pixeles. Pero a medida que se puso a disposición una mayor variedad de tamaños de pantalla y densidades de pixeles, trabajando con los píxeles se hizo indeseable para los programadores que tratan de escribir aplicaciones que se ven más o menos la misma en muchos dispositivos. se requiere otra solución.

Estas soluciones se iniciaron con los sistemas operativos para ordenadores de sobremesa y luego fueron adaptados para dispositivos móviles. Por esta razón, se ilumina para comenzar esta exploración con el escritorio.

pantallas de video de escritorio tienen una amplia gama de dimensiones en píxeles, de la casi obsoleto 640×480 en un máximo a los miles. La relación de aspecto de 4: 3 fue una vez estándar para ordenador muestra y para películas y televisión, así-pero la relación de aspecto de alta definición de 16: 9 (o lo similar 16:10) es ahora más común.

pantallas de video de escritorio también tienen una dimensión física generalmente se mide a lo largo de la diagonal de la pantalla en pulgadas o centímetros. La dimensión de píxeles combinado con la dimensión física le permite calcular la densidad de píxeles de resolución o de la pantalla de video en puntos por pulgada (ppp), a veces también se conoce como píxeles por pulgada (PPI). La resolución de pantalla también se puede medir como un tamaño de punto, que es la distancia entre los centros de los píxeles adyacentes, generalmente se mide en milímetros.

Por ejemplo, puede utilizar el teorema de Pitágoras para calcular que una antigua pantalla de 800×600 tiene una longitud diagonal de 1000, la raíz cuadrada de 800 cuadrado más de 600 cuadrado. Si este monitor tiene un 13 pulgadas de diagonal, que es una densidad de píxeles de 77 DPI, o un tamaño de punto de 0,33 milímetros. Sin embargo, una pantalla de 13 pulgadas en un ordenador portátil moderna podría tener dimensiones en píxeles de 2560×1600 , que es una densidad de píxeles de alrededor de 230 DPI, o un tamaño de punto de aproximadamente 0,11 milímetros. Un objeto cuadrado 100 píxeles en esta pantalla es un tercio del tamaño de un mismo objeto en la pantalla mayor.

Los programadores deben tener la oportunidad de luchar cuando se trata de elementos visuales de tamaño correctamente. Por esta razón, tanto Apple como Microsoft para sistemas de computación de escritorio que permiten a los programadores para trabajar con la pantalla de video en alguna forma de unidades independientes del dispositivo en lugar de píxeles idearon. La mayor parte de las dimensiones que un encuentro programador y especifica están en estas unidades independientes del dispositivo. Es la responsabilidad del sistema operativo para convertir de ida y vuelta entre estas unidades y píxeles.

En el mundo de Apple, pantallas de video de escritorio se supone tradicionalmente para tener una resolución de 72 unidades por pulgada. Este número proviene de la tipografía, donde muchas mediciones están en unidades de **puntos**. En la tipografía clásica, existen aproximadamente 72 puntos por pulgada, pero en tipografía digital el punto ha sido estandarizado para ser exactamente un setentaosavo de pulgada. Al trabajar con puntos en lugar de píxeles, un programador tiene un sentido intuitivo de la relación entre los tamaños numéricos y el área que ocupan los objetos visuales en la pantalla.

En el mundo de Windows, se ha desarrollado una técnica similar, llamado *píxeles independientes del dispositivo* (DIP) o *unidades independientes del dispositivo* (UDE). Para un programador de Windows, pantallas de video de escritorio se supone que tienen una resolución de 96 UDE, que es exactamente un tercio mayor que 72 DPI, aunque puede ser ajustado por el usuario.

Los dispositivos móviles, sin embargo, tienen algo diferentes normas: Las densidades de píxeles obtenidos en los teléfonos modernos son típicamente mucho más alta que en las pantallas de escritorio. Esta mayor densidad de píxeles permite que el texto y otros objetos visuales para reducir el tamaño mucho más en tamaño antes de convertirse en ilegible.

Los teléfonos también se llevan a cabo típicamente mucho más cerca de la cara del usuario que es una pantalla de escritorio o portátil. Esta diferencia también implica que los objetos visuales en el teléfono pueden ser más pequeños que los objetos comparables en las pantallas de escritorio o portátil. Debido a las dimensiones físicas del teléfono son mucho más pequeñas que las pantallas de escritorio, la reducción de los objetos visuales hacia abajo es muy deseable, ya que permite mucho más para que quepa en la pantalla.

Apple sigue para referirse a las unidades independientes del dispositivo en el iPhone como *puntos*. Hasta hace poco, todos de alta densidad pantallas que de Apple de Apple se refiere a por el nombre de marca Retina-tienen una conversión de dos píxeles al punto. Este fue el caso para el MacBook Pro, iPad y iPhone. La excepción es el reciente iPhone 6 Plus, que tiene tres píxeles hacia el punto.

Por ejemplo, la dimensión de píxeles 640×960 de la pantalla de 3,5 pulgadas de la iPhone 4 tiene una densidad de píxeles actual de alrededor de 320 DPI. Hay dos píxeles para el punto, por lo que a un programa de aplicación que se ejecuta en el iPhone 4, la pantalla parece tener una dimensión de 320×480 puntos. El iPhone 3 en realidad no tienen una dimensión de píxel de 320×480 , y los puntos igualado píxeles, por lo que un programa que se ejecuta en estos dos dispositivos, las pantallas del iPhone 3 y iPhone 4 parecen ser del mismo tamaño. A pesar de los mismos tamaños percibidas, objetos gráficos y de texto se muestran en una mayor resolución en el iPhone 4 que el iPhone 3.

Para el iPhone 3 y el iPhone 4, la relación entre el tamaño de pantalla y de punto dimensiones implica un factor de conversión de 160 puntos por pulgada en lugar de la estándar de escritorio de 72.

El iPhone 5 tiene una pantalla de 4 pulgadas, pero la dimensión de píxeles es 640×1136 . La densidad de píxeles es aproximadamente el mismo que el iPhone 4. Para un programa, esta pantalla tiene un tamaño de 320×768 puntos.

El iPhone 6 tiene una pantalla de 4,7 pulgadas y una dimensión de píxel de 750×1334 . La densidad de píxeles es también de 320 DPI. Hay dos píxeles hasta el punto, por lo que un programa, la pantalla parece tener un tamaño de punto de 375×667 .

Sin embargo, el iPhone 6 Plus tiene una pantalla de 5,5 pulgadas y una dimensión de píxel de 1080×1920 , que es una densidad de píxeles de 400 DPI. Esta densidad de píxeles más alto implica más píxeles hasta el punto, y para el iPhone 6 Plus, Apple ha puesto en el punto equivalente a tres píxeles. Que normalmente implica un tamaño de pantalla percibida de 360×640 puntos, pero a un programa, la pantalla iPhone 6 Plus tiene un tamaño de punto de 414×736 , por lo que la resolución percibida es de unos 150 puntos por pulgada.

Esta información se resume en la siguiente tabla:

Modelo	iPhone 2, 3	Iphone 4	iphone 5	iphone 6	iPhone 6 Plus *
tamaño de pixel	320×480	$640 \times 960 \times 640$	$1136 \times 1334 \times 1080$	1920	
diagonal de la pantalla	3,5 pulg.	3,5 pulg.	4 en.	4,7 en.	5,5 pulg.
Densidad de pixeles	165 DPI	330 DPI	326 DPI	326 DPI	401 DPI
Píxeles por punto	1	2	2	2	3
tamaño de punto	320×480	320×480	320×568	375×667	414×736
Puntos por pulgada	165	165	163	163	154

* Incluye 115 por ciento la disminución de resolución.

Android hace algo muy similar: los dispositivos Android tienen una amplia variedad de tamaños y dimensiones de los píxeles, pero un programador de Android funciona generalmente en unidades de píxeles independientes de la densidad (DPS). La relación entre los píxeles y dps se establece suponiendo 160 dps por pulgada, lo que significa que las unidades independientes del dispositivo Apple y Android son muy similares.

Microsoft tomó un enfoque diferente con Windows Phone 7. El teléfono de Windows 7 original dispositivos tenían una dimensión de pantalla de 480×800 píxeles, que a menudo se refiere como WVGA (Wide Video Graphics

Formación). Aplicaciones trabajaron con esta pantalla en unidades de píxeles. Si se supone un tamaño medio de pantalla de 4 pulgadas para un dispositivo de 480×800 Windows Phone 7, esto significa que Windows Phone 7 supone implícitamente una densidad de píxeles de alrededor de 240 DPI. Eso es 1,5 veces la densidad de píxeles supuesta de dispositivos iPhone y Android. Con el tiempo, se permitió a varias pantallas de mayor tamaño: 768×1280 (WXGA o Amplia Extended Graphics Array), 720×1280 (denominado de alta definición utilizando la jerga de la televisión como 720p), y 1080×1920 (1080p llamada). Para estos tamaños de pantalla adicionales, los programadores trabajaron en unidades independientes del dispositivo. Un factor de escala interna traducido entre los píxeles y unidades independientes del dispositivo de manera que el ancho de la pantalla en el modo vertical siempre parecía ser 480 píxeles.

Con la API de Windows en tiempo de ejecución en Windows Phone 8.1, se introdujeron diferentes factores de escala basándose tanto en tamaño de píxel de la pantalla y el tamaño físico de la pantalla. En la tabla siguiente se armó basa en el teléfono de Windows 8.1 emuladores utilizando un programa llamado **Qué tamaño**, que verá en breve:

tipo de pantalla	WVGA de 4"	WXGA de 4.5"	720p 4.7"	1080p 5.5"	1080p 6"
tamaño de pixel	480×800	768×1280	720×1280	1080×1920	1080×1920
Tamaño en UDE	400×640	$384 \times 614,5$	400×684	450×772	491×847
Factor de escala	1.2	2	1.8	2.4	2.2
DPI	194	161	169	167	167

Los factores de escala se calcula a partir de la anchura debido a la altura en UDE representada por la **Qué tamaño** programa excluye la barra de estado de Windows Phone. Las cifras DPI finales se calcularon basándose en el tamaño de píxel completo, el tamaño diagonal de la pantalla en pulgadas, y el factor de escala.

Aparte del valor atípico WVGA, el DPI calculada es lo suficientemente cerca del criterio 160 DPI asociado con los dispositivos iOS y Android.

Windows 10 Mobile utiliza factores de escala algo más altas, y en múltiplos de 0,25 en lugar de 0,2. En la siguiente tabla se elaboró sobre la base de los emuladores de Windows Mobile 10:

tipo de pantalla	WVGA de 4"	QHD de 5.2"	WXGA de 4.5"	720 5"	1080p 6"
tamaño de pixel	480×800	540×960	768×1280	720×1280	1080×1920
Tamaño en UDE	320×512	360×616	341×546	360×616	432×744
Factor de escala	1.5	1.5	2.25	2	2.5
DPI	155	141	147	147	141

Se podría concluir de ello que un buen promedio DPI para Windows 10 móvil es de 144 (redondeado al múltiplo más cercano de 16) en lugar de 160. O se podría decir que es lo suficientemente cerca de 160 a asumir que es compatible con iOS y Windows Phone.

Xamarin.Forms tiene una filosofía de la utilización de las convenciones de las plataformas subyacentes tanto como sea posible. De acuerdo con esta filosofía, un programador Xamarin.Forms trabaja con tamaños definidos por cada plataforma en particular. Todos los tamaños que el programador encuentra a través de la API Xamarin.Forms están en estas unidades específicas de la plataforma, independiente del dispositivo.

Xamarin.Forms programadores en general, pueden tratar la pantalla del teléfono en una forma independiente del dispositivo, con la siguiente resolución:

- 160 unidades por pulgada
- 64 unidades a la centímetro

los **VisualElement** clase define dos propiedades, nombrados **Anchura** y **Altura**, que proporcionan las dimensiones prestadas de puntos de vista, diseños y páginas en estas unidades independientes del dispositivo. Sin embargo, la configuración inicial de Anchura y Altura son valores "mock" de -1. Los valores de estas propiedades pasan a ser válidos sólo cuando el sistema de diseño se ha posicionado y dimensionado todo en la página. Además, tenga en cuenta que el valor predeterminado Llenar el establecimiento de **HorizontalOptions** o **VerticalOptions** a menudo causa a fin de ocupar más espacio de lo que sería de otra manera. los Anchura y Altura valores reflejan este espacio extra. los Anchura y Altura valores también incluyen cualquier Relleno que se puede fijar en el elemento y son consistentes con el área coloreada por el punto de vista de Color de fondo propiedad.

VisualElement define un evento denominado **SizeChanged** que se dispara cada vez que el Anchura o

Altura propiedad de los cambios de elementos visuales. Este evento es parte de varias notificaciones que se producen cuando una página se presenta, un proceso que involucra los diversos elementos de la página estando dimensionado y posicionado. Este proceso de disposición se produce después de la primera definición de una página (por lo general en la página constructor), y un nuevo pase disposición tiene lugar en respuesta a cualquier cambio que pudiera afectar a disposición, por ejemplo, cuando se añaden vistas a una Pagina de contenido o una StackLayout, retirados de estos objetos, o cuando las propiedades se establecen en los elementos visuales que podrían resultar en sus tamaños cambiante.

Un nuevo diseño también se desencadena cuando cambia el tamaño de la pantalla. Esto sucede sobre todo cuando el teléfono se gira entre los modos vertical y horizontal.

Un total familiaridad con el sistema Xamarin.Forms diseño a menudo acompaña el trabajo de escribir su propio Disposición <Ver> derivados. Esta tarea nos espera en el Capítulo 26, "diseños personalizados." Hasta entonces, sólo tiene que saber cuándo Anchura y Altura propiedades de cambio es muy útil para trabajar con tamaños de objetos visuales. Puede adjuntar una **SizeChanged** handler a cualquier objeto visual de la página, incluyendo la propia página. los

Qué tamaño programa muestra cómo obtener el tamaño de la página y mostrarla:

```
clase pública WhatSizePage : Pagina de contenido
{
    Etiqueta etiqueta;

    público WhatSizePage ()
    {
        etiqueta = nuevo Etiqueta
        {
            Tamaño de Letra = Dispositivo .GetNamedSize ( NamedSize .Grande, tipo de ( Etiqueta )), 
            HorizontalOptions = LayoutOptions .Centrar,
            VerticalOptions = LayoutOptions .Centrar
        };

        Content = etiqueta;

        SizeChanged += OnPageSizeChanged;
    }

    vacío OnPageSizeChanged ( objeto remitente, EventArgs args)
```

```
{  
    label.text = Cuerda .Formato( "{0} \u00D7 {1}" , Anchura, altura);  
}  
}
```

Este es el primer ejemplo de cómo tratar en este libro acontecimiento, y se puede ver que los eventos se manejan de la normal de C # y .NET manera. El código al final del constructor concede la OnPageSize-cambiado controlador de eventos al SizeChanged caso de la página. El primer argumento para el controlador de eventos (habitualmente llamado remitente) es el objeto de disparar el evento, en este caso la instancia de Qué tamaño-Página, pero el controlador de eventos que no utiliza. Tampoco utilice el controlador de eventos al segundo argumento, el llamado *argumentos del evento* -que a veces proporciona más información sobre el evento.

En su lugar, el controlador de eventos tiene acceso a la Etiqueta elemento (convenientemente guardado como un campo) para mostrar la Anchura y Altura propiedades de la página. El carácter Unicode en el String.Format llamada es un símbolo veces (x).

los SizeChanged evento no es la única oportunidad de obtener el tamaño de un elemento. VisualElement también define un método virtual protegido nombrado OnSizeAllocated que indica cuando el elemento visual se asigna un tamaño. Puede anular este método en su Pagina de contenido derivado en lugar de la manipulación del SizeChanged evento, pero OnSizeAllocated a veces se llama cuando el tamaño no está realmente cambiando.

Aquí está el programa que se ejecuta en las tres plataformas estándar:



Para el registro, éstas son las fuentes de las pantallas de estas tres imágenes:

- El simulador de iPhone 6, con dimensiones de píxeles de 750 x 1334.

- Un Nexus LG 5 con un tamaño de pantalla de 1080 x 1920 píxeles.
- Un Nokia Lumia 925 con un tamaño de pantalla de 768 x 1280 píxeles.

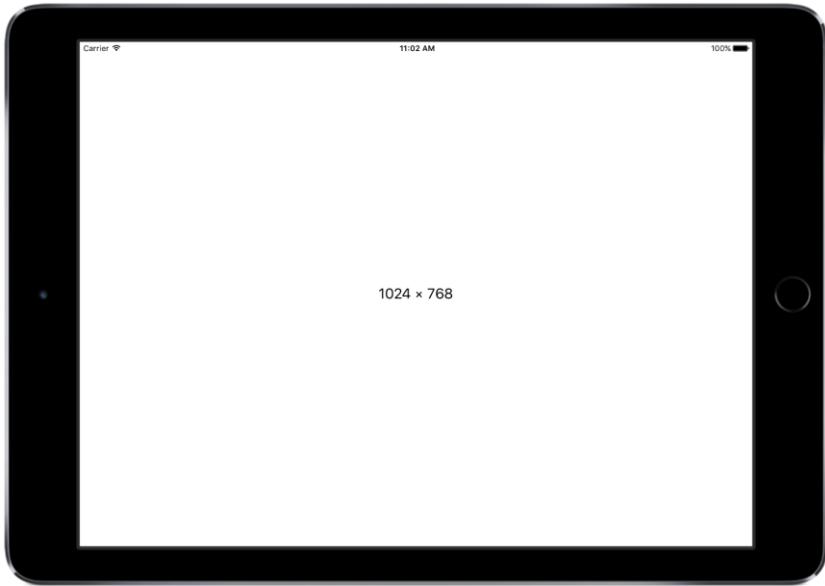
Observe que el tamaño vertical percibida por el programa en el Android no incluye el área ocupada por la barra de estado o botones inferiores; el tamaño vertical en el dispositivo móvil de Windows 10 no incluye el área ocupada por la barra de estado.

Por defecto, las tres plataformas responden a los cambios de orientación del dispositivo. Si activa los teléfonos (o emuladores) 90 grados hacia la izquierda, los móviles muestran los siguientes tamaños:



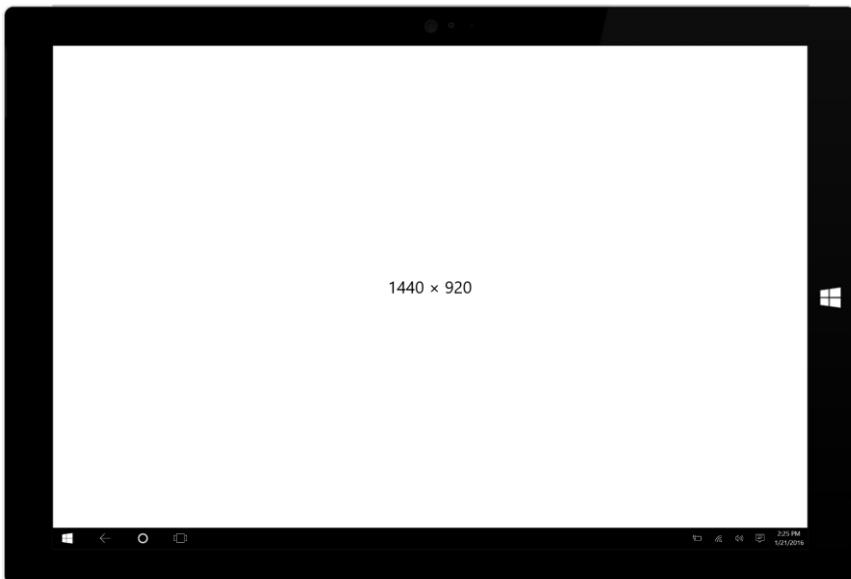
Las capturas de pantalla de este libro están diseñados sólo para el modo vertical, por lo que necesitas para convertir este libro de lado para ver lo que el programa se ve como en el paisaje. La anchura 598-pixel en el Android excluye el área para los botones; la altura de 335 píxeles excluye la barra de estado, que siempre aparece sobre la página. En el dispositivo móvil de Windows 10, la anchura de 728 píxeles excluye el área de la barra de estado, que aparece en el mismo lugar pero con iconos rotados para reflejar la nueva orientación.

Aquí está el programa que se ejecuta en el simulador de Aire 2 iPad con una dimensión de píxeles de 2048 x 1536.



Obviamente, el factor de escala es 2. La pantalla es de 9,7 pulgadas de diagonal para una resolución de 132 DPI.

La superficie Pro 3 tiene una dimensión de píxeles de 2160 × 1440. El factor de escala es seleccionable por el usuario para hacer todo en la pantalla más grande o más pequeño, pero el factor de escala recomendada es de 1,5:



La altura mostrada por **Qué tamaño** excluye la barra de tareas en la parte inferior de la pantalla. La pantalla es de 12" en diagonal para una resolución de 144 DPI.

Algunas notas sobre la **Qué tamaño** programa en sí:

Qué tamaño crea un único Etiqueta en su constructor y establece el Texto propiedad en el controlador de eventos. Eso no es la única manera de escribir un programa de este tipo. El programa podría utilizar el SizeChanged manejador para crear toda una nueva Etiqueta con el nuevo texto y establecer esa nueva Etiqueta como el contenido de la página, en cuyo caso la anterior Etiqueta se convertiría sin referencias y por lo tanto aptos para reciclaje. Pero la creación de nuevos elementos visuales es innecesario e inútil en este programa. Es mejor para el programa para crear un único Etiqueta vista y simplemente ajustar la Texto propiedad para indicar el nuevo tamaño de la página.

cambios de tamaño de monitoreo es la única manera de una aplicación Xamarin.Forms puede detectar los cambios de orientación sin obtener la información específica de la plataforma. Es la anchura mayor que la altura? Eso es paisaje. De lo contrario, es el retrato.

Por defecto, las plantillas de Visual Studio y Studio Xamarin para soluciones Xamarin.Forms permiten cambios de orientación de dispositivo para las tres plataformas. Si desea desactivar la orientación cambios, por ejemplo, si usted tiene una aplicación que simplemente no funciona bien en modo retrato o paisaje, puede hacerlo.

Para iOS, mostrar primero el contenido del Info.plist en Visual Studio o Xamarin Studio. En el **iPhone Información de despliegue** sección, utilice el **Orientaciones dispositivo compatible** área para especificar qué orientaciones se permiten.

Para Android, en el Actividad atribuir a la Actividad principal clase en el archivo MainActivity.cs, añadir:

```
screenOrientation = Orientación de la pantalla .Paisaje
```

O:

```
screenOrientation = Orientación de la pantalla .Retrato
```

los Actividad atributo generado por la plantilla de solución contiene una ConfigurationChanges argumento de que también se refiere a Orientación de la pantalla, pero el propósito de ConfigurationChanges es inhibir un reinicio de la actividad cuando la orientación o la pantalla cambia el tamaño del teléfono.

Para los dos proyectos de Windows Phone, la clase y la enumeración de utilizar se encuentra en el Windows-. Graphics.Display espacio de nombres. En el Página principal constructor en el archivo MainPage.xaml.cs, establezca la estática DisplayInformation. propiedad a uno o más miembros de la DisplayOrientations enumeración combinado con el C # bit a bit o funcionamiento. Para restringir el programa a horizontal o vertical, utilice:

```
DisplayInformation.AutoRotationPreferences = DisplayOrientations.Paisaje
```

O:

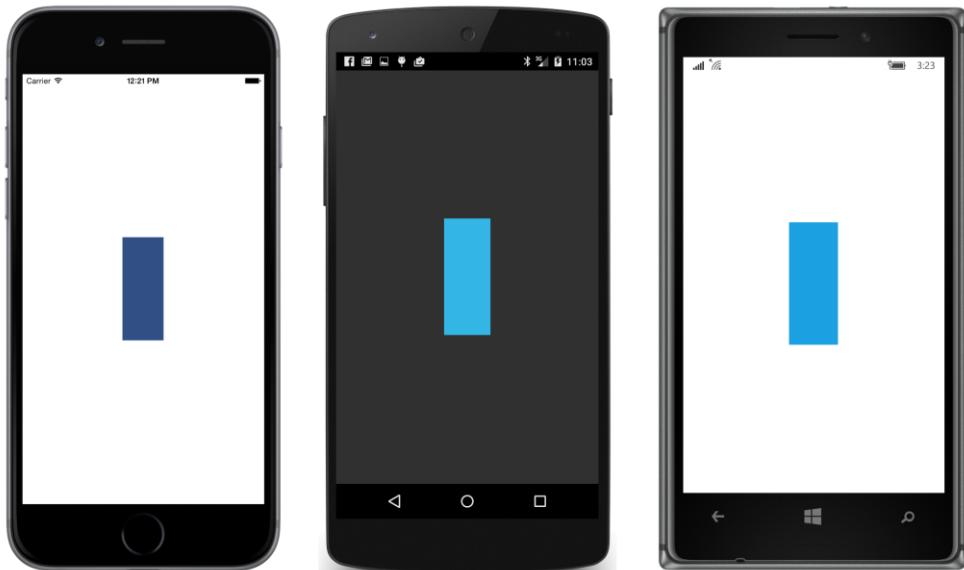
```
DisplayInformation.AutoRotationPreferences = DisplayOrientations.Retrato;
```

tamaños métricos

Ahora que ya sabe cómo tamaños en una aplicación Xamarin.Forms corresponden aproximadamente a las dimensiones métricas de pulgadas y centímetros, puede elementos de tamaño de manera que son aproximadamente del mismo tamaño en diferentes dispositivos. Aquí hay un programa llamado **MetricableViewPage** que muestra una BoxView con una anchura de aproximadamente un centímetro y una altura de aproximadamente una pulgada:

```
p\xedblico clase MetricableViewPage : Página de contenido
{
    p\xedblico MetricableViewPage ()
    {
        content = nuevo BoxView
        {
            color = Color .Acento,
            WidthRequest = 64,
            HeightRequest = 160,
            HorizontalOptions = LayoutOptions .Centrar,
            VerticalOptions = LayoutOptions .Centrar
        };
    }
}
```

Si realmente tomar una regla para el objeto en la pantalla de su teléfono, usted encontrará que no es exactamente el tamaño deseado pero ciertamente cerca de ella, ya que estas imágenes también confirman:



Este programa está destinado para funcionar en los teléfonos. Si desea ejecutarlo en tabletas, así, es posible utilizar el Device.Idiom propiedad para establecer un factor algo menor para las tabletas iPad y Windows.

tamaño de las fuentes estimadas

los **Tamaño de fuente propiedad de Etiqueta y Botón** especifica la altura aproximada de caracteres de la fuente desde el fondo de descendentes a la parte superior de los ascendentes, a menudo (dependiendo de la fuente), incluyendo marcas diacríticas así. En la mayoría de los casos usted desea establecer esta propiedad a un valor devuelto por la **Delawarevice.GetNamedSize** método. Esto le permite especificar un miembro de la **NamedSize** enumeración:

De manera predeterminada, la micro, pequeña, mediana, O Grande.

Como alternativa, se puede establecer el Tamaño de fuente propiedad a tamaño real de la fuente numéricos, pero hay un pequeño problema implicado (que se discutirá en detalle en breve). En su mayor parte, se especifican los tamaños de fuente en las mismas unidades independientes del dispositivo utilizadas a lo largo de Xamarin.Forms, lo que significa que se pueden calcular los tamaños de fuente independientes del dispositivo en base a la resolución de la plataforma.

Por ejemplo, supongamos que desea utilizar una fuente de 12 puntos en su programa. Lo primero que debe saber es que mientras que una fuente de 12 puntos podría ser de un tamaño cómodo para el material impreso o una pantalla del escritorio, en un teléfono que es bastante grande. Pero vamos a continuar.

Hay 72 puntos por pulgada, por lo que una fuente de 12 puntos es un sexto de pulgada. Multiplicar por el DPI de resolución de 160 y eso es alrededor de 27 unidades independientes del dispositivo.

Vamos a escribir un pequeño programa llamado **FontSizes**, que comienza con una pantalla similar a la **NamedFontSizes** programa en el Capítulo 3, pero a continuación, muestra un texto con tamaños de punto numéricos, se convirtió al independiente del dispositivo unidades usando la resolución del dispositivo:

```
clase pública FontSizesPage : Pagina de contenido
{
    público FontSizesPage ()
    {
        BackgroundColor = Color .Blanco;
        StackLayout stackLayout = nuevo StackLayout
        {
            HorizontalOptions = LayoutOptions .Centrar,
            VerticalOptions = LayoutOptions .Centrar
        };

        // Hacer los valores NamedSize.
        NamedSize [] NamedSizes =
        {
            NamedSize .Defecto, NamedSize .Micro, NamedSize .Pequeña,
            NamedSize .Medio, NamedSize .Grande
        };

        para cada ( NamedSize namedSize en namedSizes )
        {
            doble fontSize = Dispositivo .GetNamedSize ( namedSize, tipo de ( Etiqueta ));

            stackLayout.Children.Add ( nuevo Etiqueta
            {
                text = Cuerda .Formato( "Tamaño Named = {0} ({1: F2})" ,

```

```

        namedSize, fontSize),
        FontSize = fontSize,
        TextColor = Color.Negro
    });
}

// Resolución en unidades independientes del dispositivo por pulgada.
doble resolución = 160;

// Dibujar linea de separación horizontal.
stackLayout.Children.Add (
    nuevo BoxView
    {
        color = Color.Acento,
        HeightRequest = resolución / 80
    });
}

// Hacer algunos tamaños de punto numéricos.
En t [] PtSizes = {4, 6, 8, 10, 12};

para cada ( doble ptSize en ptSizes)
{
    doble fontSize = resolución * ptSize / 72;

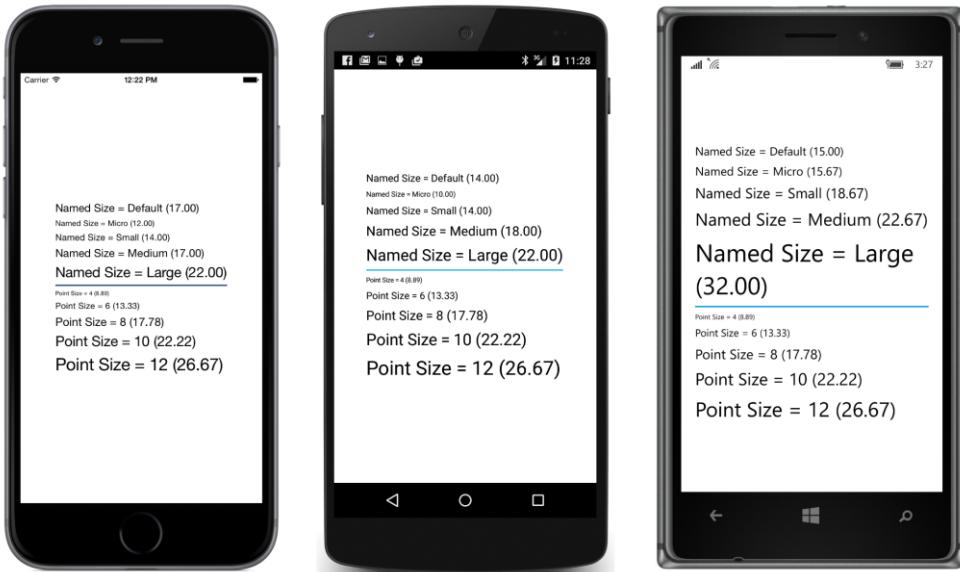
    stackLayout.Children.Add ( nuevo Etiqueta
    {
        text = Cuerda .Formato("Tamaño de punto = {0} ({1: F2})", 
            ptSize, fontSize),
        FontSize = fontSize,
        TextColor = Color.Negro
    });
}

Content = stackLayout;
}
}

```

Para facilitar las comparaciones entre las tres pantallas, los fondos se han establecido de manera uniforme a blanco y negro a las etiquetas. Observe la BoxView insertado en el StackLayout entre los dos para cada bloques: la HeightRequest ponele le da una altura independiente del dispositivo de aproximadamente un ochenta de pulgada, y se asemeja a una línea horizontal.

Curiosamente, los tamaños visuales resultantes basados en el cálculo son más consistentes entre las plataformas que los tamaños mencionados. Los números entre paréntesis son los numérica Tamaño de fuente valores en unidades independiente del dispositivo:



ajuste de texto al tamaño disponibles

Puede que sea necesario para adaptarse a un bloque de texto a un área rectangular en particular. Es posible calcular un valor para la Tamaño de fuente propiedad de Etiqueta basado en el número de caracteres de texto, el tamaño del área rectangular, y sólo dos números.

El primer número es espacio de línea. Esta es la altura vertical de una Etiqueta Ver por línea de texto. Para las fuentes predeterminadas asociadas con las tres plataformas, es más o menos relacionada con la Tamaño de fuente propiedad de la siguiente manera:

- **iOS:** interlineado = 1.2 * label.FontSize
- **Androide:** interlineado = 1.2 * label.FontSize
- **Telefono windows:** interlineado = 1.3 * label.FontSize

El segundo número es útil ancho de carácter promedio. Para una mezcla normal de las letras mayúsculas y minúsculas para las fuentes por defecto, este ancho de carácter promedio es aproximadamente la mitad del tamaño de la fuente, independientemente de la plataforma:

- averageCharacterWidth = 0,5 * label.FontSize

Por ejemplo, supongamos que desea para adaptarse a una cadena de texto que contiene 80 caracteres en un ancho de 320 unidades, y desea que el tamaño de la fuente sea lo más grande posible. Divida la anchura (320) por medio del número de caracteres (40), y se obtiene un tamaño de fuente de 8, que se puede fijar a la Tamaño de fuente propiedad de Etiqueta. por

texto que es un tanto indeterminada y no puede ser probado de antemano, es posible que desee hacer este cálculo un poco más conservador para evitar sorpresas.

El siguiente programa utiliza tanto espacio entre líneas y el ancho promedio de carácter para adaptarse a un párrafo de texto en la página, menos el área en la parte superior del iPhone ocupada por la barra de estado. Para hacer que la exclusión de la condición de iOS bar un poco más fácil en este programa, el programa utiliza una ContentView.

ContentView deriva de Diseño pero sólo añade una Contenido propiedad a lo que se hereda de Diseño. ContentView también es la clase base para Marco. A pesar de que ContentView no tiene ninguna funcionalidad que no sea ocupando un área rectangular de espacio, que es útil para dos propósitos: Muy a menudo, Contenido-Ver puede ser un parente a otras vistas a definir una nueva vista personalizada. Pero ContentView También puede simular un margen.

Como habrá notado, Xamarin.Forms no tiene concepto de un margen, que tradicionalmente es similar a la de relleno, excepto que el relleno se encuentra dentro de una visión y una parte de la vista, mientras que el margen se encuentra fuera de la vista y en realidad parte de vista de los padres . UN ContentView nos permite simular este. Si usted encuentra la necesidad de establecer un margen en una vista, poner la vista en una ContentView y establecer el Relleno propiedad en el Estafa-

tentView. ContentView hereda una Relleno propiedad de Diseño.

los **EstimatedFontSize** usos del programa ContentView de una manera ligeramente diferente: Se establece el relleno habitual en la página para evitar la barra de estado iOS, pero entonces se establece un ContentView como el contenido de esa página. Por lo tanto, este ContentView es del mismo tamaño que la página, pero excluyendo la barra de estado IOS. Es en este ContentView que el **SizeChanged** evento está unido, y es el tamaño de esta Estafa-

tentView que se utiliza para calcular el tamaño de fuente del texto.

los **SizeChanged** manejador utiliza el primer argumento para obtener el objeto de disparar el evento (en este caso el ContentView), que es el objeto en el que la Etiqueta debe encajar. El cálculo se describe en los comentarios:

```
público clase EstimatedFontSizePage : Pagina de contenido
{
    Etiqueta etiqueta;

    público EstimatedFontSizePage ()
    {
        etiqueta = nuevo Etiqueta ();
        Relleno = nuevo Espesor (0, Dispositivo .OnPlatform (20, 0, 0), 0, 0);
        ContentView contentView = nuevo ContentView
        {
            Content = etiqueta
        };
        contentView.SizeChanged += OnContentSizeChanged;
        Content = contentView;
    }

    vacío OnContentSizeChanged ( objeto remitente, EventArgs args )
    {
        cuerda text =
            "Una fuente predeterminada del sistema con un tamaño de fuente de S" +

```

```

    "Tiene una altura de línea de alrededor de ((0: F1) * S) y un" +
    "Ancho de carácter promedio de alrededor de ((1: F1) * S)." +
    "En esta página, que tiene una anchura de (2: F0) y un" +
    "Altura de (3: F0), un tamaño de fuente de 1 debería" +
    "Cómodamente hacer que el ?? 2 personajes de esta" +
    "Con el párrafo? 3 líneas y aproximadamente 4 caracteres?" +
    "Por línea. ¿Funciona?";

    // Obtener vista cuyo tamaño está cambiando.
    Ver view = (Ver)remitente;

    // Definir dos valores como múltiplos de tamaño de fuente.
    doble lineHeight = Dispositivo.OnPlatform(1.2, 1.2, 1.3);
    doble charWidth = 0.5;

    // formato al texto y obtener la longitud de caracteres.
    text = Cuerda.formato(texto, lineHeight, charWidth, view.Width, view.Height);
    En t charCount = text.length;

    // Porque:
    //      LineCount = view.Height / (lineHeight * fontSize)
    //      charsPerLine = view.Width / (charWidth * fontSize)
    //      charCount = LineCount * charsPerLine
    // Por lo tanto, la solución para fontSize:
    En t fontSize = (En t) Mates.Sqrt(view.Width * view.Height /
        (CharCount * lineHeight * charWidth));

    // Ahora estos valores pueden ser calculados.
    En t LineCount = (En t)(View.Height / (lineHeight * fontSize));
    En t charsPerLine = (En t)(View.Width / (charWidth * fontSize));

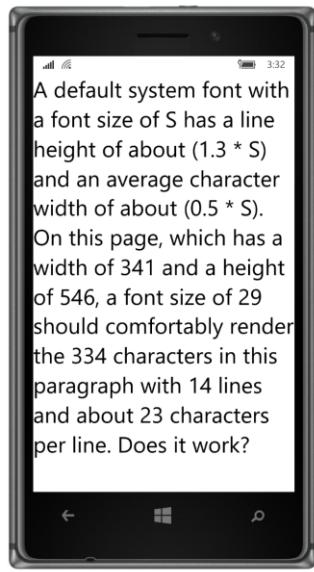
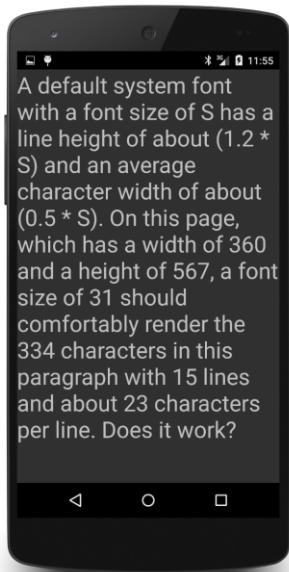
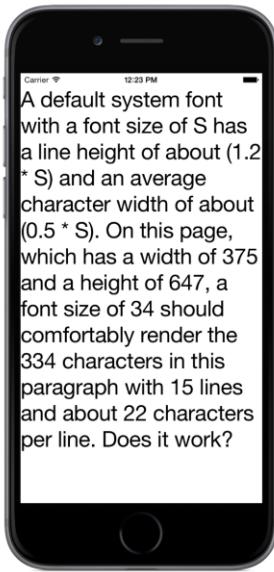
    // Reemplazar los marcadores de posición con los valores.
    text = text.Replace("?", 1, fontSize.ToString());
    text = text.Replace("?", 2, CharCount.ToString());
    text = text.Replace("?", 3, LineCount.ToString());
    text = text.Replace("?", 4, CharsPerLine.ToString());

    // Establecer las propiedades de la etiqueta.
    label.text = texto;
    label.fontSize = fontSize;
}

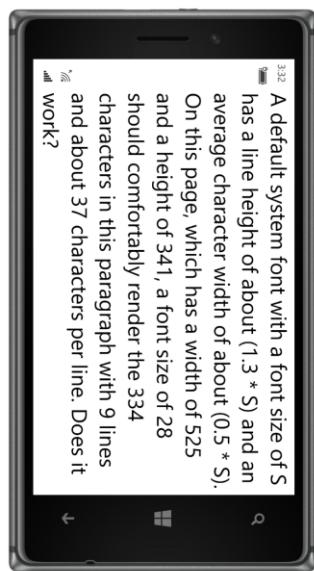
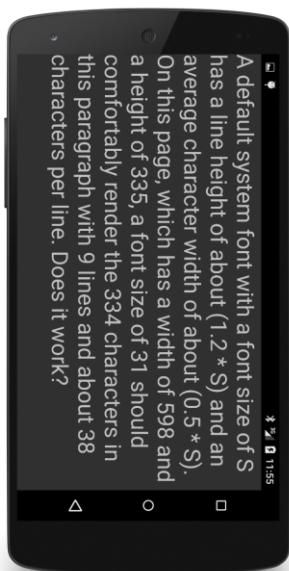
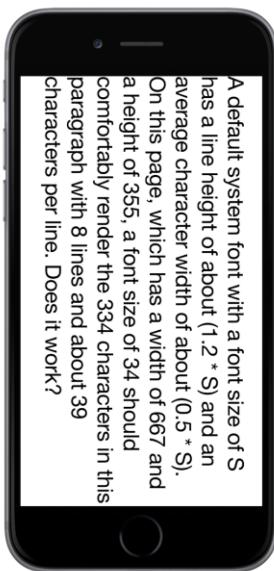
}

Los marcadores de posición de texto denominados "? 1", "? 2", "?", Y 3 "? 4" fueron elegidos para ser único, sino también a ser el mismo número de caracteres como los números que los sustituyen.

Si el objetivo es hacer que el texto lo más grande posible sin el texto se derrame fuera de la página, los resultados validan el enfoque:
```



No está mal. No está mal. El texto muestra realmente en una línea menos que se indica en las tres plataformas, pero la técnica parece sonido. No siempre es el caso de que la misma Tamaño de fuente se calcula para el modo horizontal, pero a veces sucede:



Un reloj de ajuste a tamaño

los Dispositivo clase incluye un estático `startTimer` método que le permite establecer un temporizador que dispara un evento periódico. La disponibilidad de un evento de temporizador significa que una aplicación de reloj es posible, incluso si se muestra el tiempo sólo en el texto.

El primer argumento de `Device.StartTimer` es un intervalo expresa como una Espacio de tiempo valor. El temporizador activa un suceso periódicamente en base a ese intervalo. (Se puede bajar tan bajo como 15 o 16 milisegundos, que es aproximadamente el período de la frecuencia de cuadros de 60 cuadros por segundo común en las pantallas de video.) El controlador de eventos no tiene argumentos, pero debe volver cierto para mantener el temporizador en marcha.

los `FitToSizeClock` programa crea una Etiqueta para la visualización de la hora y luego establece dos eventos: la `SizeChanged` evento en la página para cambiar el tamaño de fuente y la `Device.StartTimer` evento para intervalos de un segundo para cambiar el Texto propiedad.

Muchos programadores de C # en estos días como para definir pequeños controladores de eventos como funciones lambda anónimos. Esto permite que el código de control de eventos para estar muy cerca de la creación de instancias y la inicialización del objeto de disparar el evento en lugar de algún otro sitio en el archivo. También permite hacer referencia a los objetos dentro del controlador de eventos sin almacenar esos objetos como campos.

En este programa, los dos controladores de eventos simplemente cambiar una propiedad de la Etiqueta, y ambos se expresan como funciones lambda para que puedan tener acceso a la Etiqueta sin que se almacena como un campo:

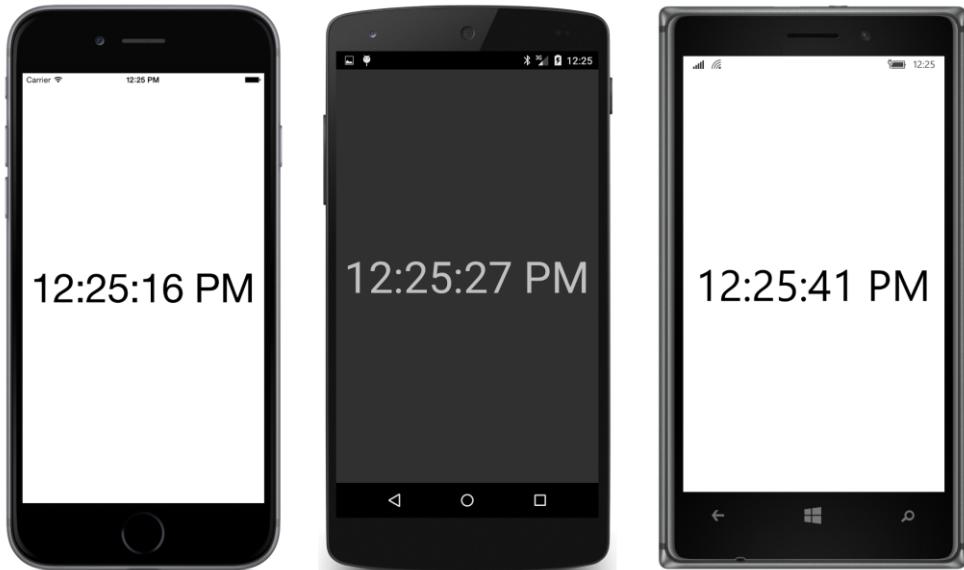
```
clase pública FitToSizeClockPage : Página de contenido
{
    público FitToSizeClockPage ()
    {
        Etiqueta clockLabel = nuevo Etiqueta
        {
            HorizontalOptions = LayoutOptions.Centrar,
            VerticalOptions = LayoutOptions.Centrar
        };
        Content = clockLabel;

        // Procesar el evento SizeChanged para la página.
        SizeChanged += (objeto remitente, EventArgs args) =>
        {
            // escalar el tamaño de fuente para el ancho de página
            // (Basado en 11 caracteres de la cadena que se muestra).
            Si (esta.Width > 0)
                clockLabel.FontSize = esta.Width / 6;
        };

        // Inicia el temporizador en marcha.
        Dispositivo.StartTimer (Espacio de tiempo.FromSeconds (1), () =>
        {
            // Establecer la propiedad de texto de la etiqueta.
            clockLabel.Text = Fecha y hora.Now.ToString ("H: mm: ss tt");
        });
    }
}
```

```
    return true ;  
});  
}  
}
```

los startTimer manejador especifica una cadena de formato personalizado de Fecha y hora que se traduce en 10 o 11 caracteres, pero dos de ellos son letras mayúsculas y los caracteres son más anchas que la media. los SizeChanged manejador asume implicitamente que 12 caracteres se muestran ajustando el tamaño de la fuente a un sexto de la anchura de la página:



Por supuesto, el texto es mucho mayor en el modo horizontal:



Esto de un segundo temporizador no marque exactamente al comienzo de cada segundo, por lo que el tiempo que se muestra puede no estar de acuerdo precisamente con otros indicadores de tiempo en el mismo dispositivo. Puede que sea más precisa mediante el establecimiento de un temporizador de intervalo más frecuente. El rendimiento no se verá afectada tanto por la pantalla sigue siendo sólo cambia una vez por segundo y no requerirá un nuevo ciclo de diseño hasta entonces.

Los problemas de accesibilidad

los **EstimatedFontSize** programa y el **FitToSizeClock** programa de ambos tienen un defecto sutil, pero el problema podría no ser tan sutil que si eres una de las muchas personas que no pueden leer cómodamente el texto en un dispositivo móvil y utiliza las funciones de accesibilidad del dispositivo para hacer el texto más grande.

En iOS, ejecute el **Ajustes** aplicación y seleccione **General**, y **Accesibilidad**, y **Texto más grande**. A continuación, puede utilizar un control deslizante para que el texto en la pantalla más grande o más pequeño. La página indica que el texto sólo se puede ajustar en las aplicaciones de iOS que apoyan la **Tipo dinámico** característica.

En Android, ejecute el **Ajustes** aplicación y seleccione **Monitor** y entonces **Tamaño de fuente**. Se le presenta con cuatro botones de radio para seleccionar **Pequeño**, **Normal** (el valor por defecto), **Grande**, o **Enorme**.

En un dispositivo móvil de Windows 10, ejecute el **Ajustes** aplicación y seleccione **Facilidad de Acceso** y entonces **Más opciones**. A continuación, puede mover un control deslizante etiquetado **Ajuste de texto** de 100% a 200%.

Esto es lo que descubrirá:

La configuración de iOS no tiene efecto sobre las aplicaciones Xamarin.Forms.

El ajuste **Android** afecta a los valores devueltos desde `Device.GetNamedSize`. Si selecciona algo distinto **Normal** y ejecutar el **FontSize**s programa de nuevo, verá que para el `NamedSize.Default` argumento, `Device.GetNamedSize` devuelve 14 cuando el ajuste es **normal** (con como la captura de pantalla anterior muestra), pero devuelve 12 para un entorno de **Pequeña**, 16 para **Grande**, y 18 1/3 para **Enorme**.

También, *todas* el texto que aparece en la pantalla de Android es un tamaño diferente, ya sea menor o mayor dependiendo de la configuración que ha seleccionado para la constante de equilibrio Tamaño de fuente valores.

En Windows 10 móvil, los valores devueltos por `Device.GetNamedSize` no dependen de la configuración de accesibilidad, pero todo el texto se muestra más grande.

Esto significa que el **EstimatedFontSize** o **FitToSizeClock** los programas no se ejecutan correctamente en Android o Windows Mobile 10 con el ajuste de la accesibilidad para ampliar el texto. Parte del texto se trunca.

Vamos a explorar esto un poco más. Los **AccessibilityTest** programa muestra dos Etiqueta elementos en su página. El primero tiene una constante Tamaño de fuente de 20, y el segundo simplemente muestra el tamaño de la primera Etiqueta cuando cambia su tamaño:

```
pública clase AccessibilityTestPage : Página de contenido
{
    pública AccessibilityTestPage ()
    {
        Etiqueta testLabel = nuevo Etiqueta
        {
            text = "FontSize de 20" + Ambiente.NewLine + "20 caracteres a través de",
            FontSize = 20,
            HorizontalTextAlignment = Alineación del texto.Centrar,
            HorizontalOptions = LayoutOptions.Centrar,
            VerticalOptions = LayoutOptions.CenterAndExpand
        };

        Etiqueta displayLabel = nuevo Etiqueta
        {
            HorizontalOptions = LayoutOptions.Centrar,
            VerticalOptions = LayoutOptions.CenterAndExpand
        };

        testLabel.SizeChanged += (remitente, args) =>
        {
            displayLabel.Text = Cuerda.Formato("{0:F0}\{1\u00D7:F0}", TestLabel.Width,
                testLabel.Height);
        };
    }

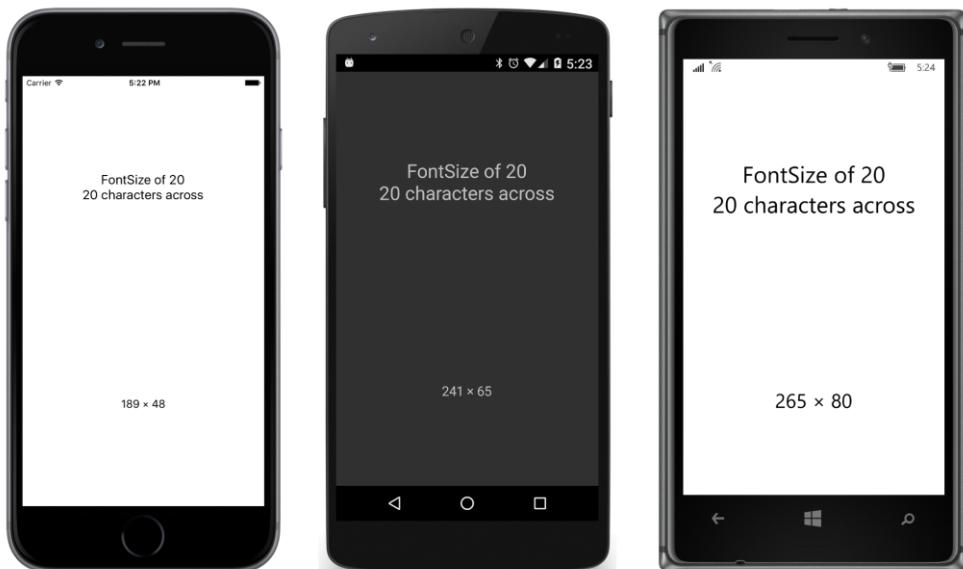
    content = nuevo StackLayout
    {
        Los niños =
        {
            testLabel,
            displayLabel
        }
    }
}
```

{}

Normalmente, el segundo Etiqueta muestra un tamaño que es más o menos consistente con los supuestos descritos anteriormente:



Pero ahora entrar en la configuración de accesibilidad y les manivela todo el camino hacia arriba. Tanto en Android y Windows Mobile pantalla de texto 10 más grande:



Los supuestos Tamaño de caracteres descritos anteriormente ya no son válidas, y por eso los programas fallan para ajustar el texto.

Pero hay un enfoque alternativo al dimensionamiento de texto a un área rectangular.

texto Empíricamente apropiado

Otro enfoque para ajustar texto dentro de un rectángulo de un tamaño particular implica empíricamente para determinar el tamaño del texto prestados basado en un tamaño de fuente en particular y luego ajustar el tamaño de fuente que arriba o hacia abajo. Este enfoque tiene la ventaja de trabajar en todos los dispositivos, independientemente de la configuración de accesibilidad.

Sin embargo, el proceso puede ser complicado: El primer problema es que no hay una relación lineal limpia entre el tamaño de la fuente y la altura del texto representado. Como texto se hace más grande respecto a la anchura de su contenedor, más saltos de línea resultan, con más espacio desperdiciado. Un cálculo para encontrar el tamaño óptimo de la fuente a menudo implica un bucle que se estrecha en el valor.

Un segundo problema está relacionado con el mecanismo real de obtener el tamaño de una Etiqueta prestado con un tamaño de fuente particular. Se puede establecer una `SizeChanged` manejador en el `Etiqueta`, pero dentro de ese manejador que no desea realizar ningún cambio (como el establecimiento de un nuevo Tamaño de fuente propiedad) que hará que las llamadas recursivas a ese controlador.

Un mejor enfoque está llamando a la `GetSizeRequest` método definido por `VisualElement` y heredado por `Etiqueta` y todos los otros puntos de vista. `GetSizeRequest` requiere dos argumentos de una restricción de la anchura y una restricción de altura. Estos valores indican el tamaño del rectángulo en el que desea guardar el elemento, y uno o el otro puede ser infinito. Cuando usas `GetSizeRequest` con un `Etiqueta`, en general, se establece el argumento de los límites de anchura a la anchura del contenedor y la restricción de altura para

`Double.PositiveInfinity`.

los `GetSizeRequest` método devuelve un valor de tipo `SizeRequest`, una estructura con dos propiedades, llamada `Solicitud` y `Mínimo`, ambos de tipo `Tamaño`. Los `Solicitud` propiedad indica el tamaño del texto representado. (Más información sobre este y otros métodos se puede encontrar en el capítulo 26.)

los `EmpiricalFontSize` proyecto demuestra esta técnica. Para mayor comodidad, se define una pequeña estructura llamada `FontCalc` cuyo constructor hace la llamada a `GetSizeRequest` para un particular, `etiqueta` (que ya está iniciado con el texto), un tamaño de fuente de prueba, y una anchura de texto:

```
struct FontCalc
{
    público FontCalc ( Etiqueta etiqueta, doble tamaño de fuente, doble containerWidth )
        : esta ()
    {
        // Guardar el tamaño de la fuente.
        FontSize = fontSize;

        // Volver a calcular la altura de etiquetas.
    }
};
```

```

label.FontSize = fontSize;
SizeRequest sizeRequest =
    label.GetSizeRequest(containerWidth, Double.PositiveInfinity);

// Guarda esa altura.
TextHeight = sizeRequest.Request.Height;
}

pública doble Tamaño de fuente { conjunto privado ; obtener ; }

pública doble Altura del texto { conjunto privado ; obtener ; }
}

```

La altura resultante de la mostrada Etiqueta se guarda en el Altura del texto propiedad.

Cuando se realiza una llamada a GetSizeRequest en una página o un diseño, la página o el diseño necesario para obtener el tamaño de todos sus hijos a través del árbol visual. Esto tiene una penalización de rendimiento, por supuesto, por lo que debe evitar hacer llamadas como que a menos que sea necesario. Sin embargo, una Etiqueta no tiene hijos, así que llamar

GetSizeRequest en un Etiqueta no es tan malo. Sin embargo, aún debe tratar de optimizar las llamadas. Evitar bucle a través de una serie secuencial de valores de tamaño de fuente para determinar el valor máximo que no resulte en el texto superior a la altura del recipiente.

Un proceso que se estrecha de forma algorítmica en el valor óptimo es mejor.

GetSizeRequest requiere que el elemento sea parte de un árbol visual y que el proceso de diseño ha comenzado, al menos parcialmente. No llame GetSizeRequest en el constructor de la clase de página. Usted no recibirá la información de ella. La primera oportunidad razonable está en una anulación de la página de OnAppearing método. Por supuesto, puede que no tenga la información suficiente en este momento para pasar argumentos al GetSizeRequest método.

Sin embargo, llamar GetSizeRequest no tiene ningún efecto secundario. No causa un nuevo tamaño que se encuentra en el elemento, lo que significa que no causa una SizeChanged evento que se disparó, lo que significa que es seguro que llamar a un SizeChanged entrenador de animales.

los EmpiricalFontSizePage crea una instancia de la clase FontCalc valores en el SizeChanged manejador de la ContentView que aloja el Etiqueta. El constructor de cada FontCalc marcas de valor GetSize-
Solicitud pide a la Etiqueta y guarda el resultante Altura del texto. los SizeChanged manejador comienza con tamaños de fuente de prueba de 10 y 100 bajo el supuesto de que el valor óptimo está en algún lugar entre estos dos y que estos representan límites inferior y superior. De ahí los nombres de las variables inferior-
FontCalc y upperFontCalc:

```

clase pública EmpiricalFontSizePage : Pagina de contenido
{
    Etiqueta etiqueta;

    público EmpiricalFontSizePage ()
    {
        etiqueta = nuevo Etiqueta ();
        Relleno = nuevo Espesor (0, Dispositivo.OnPlatform (20, 0, 0), 0, 0);
    }
}

```

```
ContentView contentView = nuevo ContentView
{
    Content = etiqueta
};

contentView.SizeChanged += OnContentSizeChanged;
Content = contentView;
}

void OnContentSizeChanged ( objeto remitente, EventArgs args)
{
    // Obtener vista cuyo tamaño está cambiando.
    Ver view = ( Ver )remitente;

    Si (view.Width <= 0 || view.Height <= 0)
        regreso ;

    label.text =
        "Este es un párrafo de texto que se muestra con" +
        "Un valor de Tamaño de Letra ?? que es empíricamente" +
        "Calculado en un bucle dentro de la SizeChanged" +
        "Manipulador de contenedores de la etiqueta. Esta técnica" +
        "Puede ser complicado: Usted no quiere entrar en" +
        "Un bucle infinito mediante la activación de un pase de disposición" +
        "Con cada cálculo. ¿Funciona?" ;

    // Calcular la altura del texto representado.
    FontCalc lowerFontCalc = nuevo FontCalc (Etiqueta, 10, view.Width);
    FontCalc upperFontCalc = nuevo FontCalc (Etiqueta, 100, view.Width);

    mientras (UpperFontCalc.FontSize - lowerFontCalc.FontSize > 1)
    {
        // Obtener el tamaño medio de la fuente de los límites superior e inferior.
        doble fontSize = (lowerFontCalc.FontSize + upperFontCalc.FontSize) / 2;

        // Comprobar la nueva altura del texto en contra de la altura del recipiente.
        FontCalc newFontCalc = nuevo FontCalc (Etiqueta, fontSize, view.Width);

        Si (NewFontCalc.TextHeight > view.Height)
        {
            upperFontCalc = newFontCalc;
        }
        más
        {
            lowerFontCalc = newFontCalc;
        }
    }

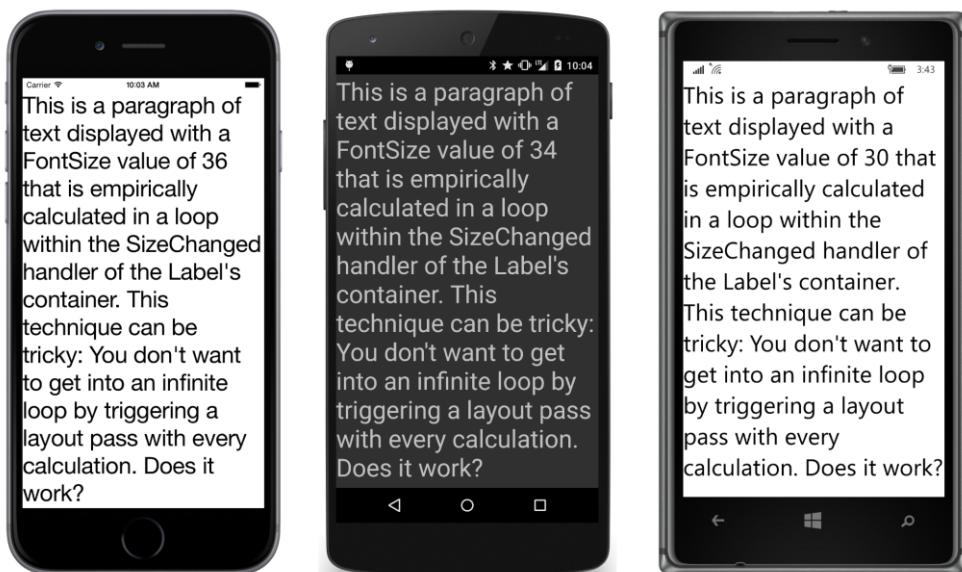
    // Establecer el tamaño de la fuente final y el texto con el valor implícito.
    label.FontSize = lowerFontCalc.FontSize;
    label.text = label.Text.Replace ("??", Label.FontSize.ToString ("F0"));
}

}
```

En cada iteración del bucle, el Tamaño de fuente propiedades de los dos FontCalc valores se promedian y un nuevo FontCalc es obtenido. Esto se convierte en el nuevo lowerFontCalc o upperFontCalc.

valor que depende de la altura del texto representado. El bucle termina cuando el tamaño de la fuente calculada está dentro de una unidad del valor óptimo.

Alrededor de siete iteraciones del bucle son suficientes para obtener un valor que es claramente mejor que el valor estimado calculado en el programa anterior:



Al girar el lado de teléfono desencadena otro nuevo cálculo que resulta en un similar (aunque no necesariamente la misma) tamaño de la fuente:



Podría parecer que el algoritmo se puede mejorar más allá de simplemente el promedio Tamaño de fuente inmuebles de la inferior y superior FontCalc valores. Pero la relación entre el tamaño de la fuente y la altura de texto representado es bastante complejo, ya veces el método más sencillo es igual de bueno.

Capítulo 6

pulsaciones de botón

Los componentes de una interfaz gráfica de usuario se pueden dividir a grandes rasgos en vistas que se utilizan para su presentación, que muestran información a el usuario, y la interacción, que obtienen entrada de el usuario. Mientras que la Etiqueta es la vista más básica presentación, el Botón es probablemente la vista interactiva arquetípico. los Botón señala un comando. Es el camino del usuario de decirle al programa para iniciar alguna acción para hacer algo.

Un Xamarin.Forms botón muestra el texto, con o sin una imagen que lo acompaña. (Sólo se mostrarán los botones de texto se describen en este capítulo; añadir una imagen a un botón se trata en el capítulo 13, "Los mapas de bits.") Cuando prensas de los dedos del usuario sobre un botón, el botón cambia de aspecto un tanto para proporcionar información al usuario. Cuando se suelta el dedo, el botón dispara una hecho clic evento. Los dos argumentos de la hecho clic manejador son típicos de Xamarin.Forms controladores de eventos:

- El primer argumento es el objeto de disparar el evento. Para el hecho clic manejador, este es el particular, Botón objeto que ha sido intervenido.
- El segundo argumento a veces proporciona más información sobre el evento. Para el hecho clic evento, el segundo argumento es simplemente una EventArgs objeto que no proporciona ninguna información adicional.

Una vez que una aplicación empieza a implementar la interacción del usuario, surgen algunas necesidades especiales: La aplicación debe hacer un esfuerzo para guardar el resultado de que la interacción si el programa pasa a ser terminado antes de que el usuario ha terminado de trabajar con él. Por esa razón, este capítulo también discute cómo una aplicación puede guardar los datos transitorios, particularmente en el contexto de los eventos del ciclo de vida de aplicaciones. Estos se describen en la sección "Almacenamiento de datos transitorios."

El procesamiento de la clic

He aquí un programa llamado **ButtonLogger** con un Botón que comparte una StackLayout con un Voluntario que contiene otro StackLayout. Cada vez que el Botón se hace clic, el programa añade un nuevo Etiqueta al desplazable StackLayout, en efecto, se registran todos los clics de los botones:

```
clase pública ButtonLoggerPage : Pagina de contenido
{
    StackLayout loggerLayout = nuevo StackLayout ();

    público ButtonLoggerPage ()
    {
        // crear el botón y adjuntar manejador se hace clic.
        Botón botón = nuevo Botón
```

```
{  
    text = "Log Haga clic en el Tiempo"  
};  
button.Clicked += OnButtonClicked;  
  
esta.Padding = nuevo Espesor (5, Dispositivo.OnPlatform (20, 0, 0), 5, 0);  
  
// Montar la página.  
esta.EsteContenido = nuevo StackLayout  
{  
    LosNiños =  
    {  
        botón,  
        nuevo ScrollView  
        {  
            VerticalOptions = LayoutOptions.FillAndExpand,  
            Content = loggerLayout  
        }  
    }  
};  
};  
  
vacío OnButtonClicked ( objeto remitente, EventArgs args)  
{  
    // Añadir etiqueta para desplazable StackLayout.  
    loggerLayout.Children.Add ( nuevo Etiqueta  
    {  
        text = "Botón hace clic en" + Fecha y hora.Now.ToString ("T")  
    });  
}  
}
```

En los programas de este libro, los controladores de eventos se dan nombres que comienzan con la palabra `En`, seguido por algún tipo de identificación de la vista de disparar el evento (a veces sólo el tipo de vista), seguido por el nombre del evento. El nombre resultante en este caso es `OnButtonClicked`.

El constructor concede la hecho clic manejador a la Botón justo después de la Botón es creado. La página se ensambla con una `StackLayout` que contiene el Botón y una `ScrollView` con otro

`StackLayout`, llamado `loggerLayout`. Observe que el `ScrollView` tiene su `VerticalOptions` ajustado a `FillAndExpand` de modo que pueda compartir el `StackLayout` con el Botón y aún así ser visible y desplazable.

Aquí está la pantalla después de que varios Botón clics:



Como se puede ver, el Botón se ve un poco diferente en las tres pantallas. Eso es debido a que el botón se representa de forma nativa en las plataformas individuales: en el iPhone es una UIButton, en Android es un Android Botón, y en Windows Mobile 10 es un tiempo de ejecución de Windows Botón. Por defecto, el botón siempre llena el área disponible para el mismo y centra el texto dentro.

Botón define varias propiedades que le permiten personalizar su apariencia:

- Familia tipográfica de tipo cuerda
- Tamaño de fuente de tipo doble
- FontAttributes de tipo FontAttributes
- Color de texto de tipo Color (por defecto es Color.Default)
- Color del borde de tipo Color (por defecto es Color.Default)
- Ancho del borde de tipo doble (por defecto es 0)
- BorderRadius de tipo doble (por defecto es 5)
- imagen (a ser discutido en el capítulo 13)

Botón También hereda la Color de fondo propiedad (y un montón de otras propiedades) de VisualElement y hereda HorizontalOptions y VerticalOptions de Ver.

Algunos Botón propiedades podrían funcionar de manera diferente en las distintas plataformas. Como se puede ver, ninguno de los botones en las capturas de pantalla tiene un borde. (Sin embargo, el botón de Windows Phone 8.1 tiene un borde blanco visible por defecto.) Si se establece el Ancho del borde propiedad en un valor distinto de cero, la frontera

se hace visible sólo en el iPhone, y es negro. Si se establece el Color del borde propiedad a algo que no sea Color.Default, la frontera es visible sólo en el dispositivo móvil de Windows 10. Si desea dejar un borde visible en ambos iOS y Windows 10 dispositivos móviles, establecer tanto Ancho del borde y Color del borde.

Pero la frontera todavía no se mostrará en los dispositivos Android a menos que también establece la Color de fondo propiedad. Personalización de un botón de frontera es una buena oportunidad para usar Device.OnPlatform (como se verá en el capítulo 10, "Extensões de marcado XAML").

los BorderRadius la propiedad tiene la intención de redondear las esquinas agudas de la frontera, y funciona en iOS y Android si se visualiza la frontera, pero no funciona en Windows 10 y Windows 10 móvil. los BorderRadius Funciona en Windows 8.1 y Windows Phone 8.1, pero si se utiliza con Color de fondo, el fondo no está encerrado dentro de la frontera.

Supongamos que escribió un programa similar al ButtonLogger pero no guardar el loggerLayout objeto como un campo. ¿Podría tener acceso a esa StackLayout objeto en el hecho clic ¿controlador de eventos?

¡Sí! Es posible obtener de padres e hijos los elementos visuales mediante una técnica llamada *recorrer el árbol visual*. Los remitente argumento de la OnButtonClicked manejador es el objeto de disparar el evento, en este caso el Botón, para que pueda comenzar el hecho clic manejador echando ese argumento:

```
Botón botón = ( Botón )remitente;
```

Usted sabe que la Botón es un hijo de una StackLayout, de modo que objeto es accesible desde el Padre propiedad. Una vez más, se requiere alguna de calidad:

```
StackLayout outerLayout = ( StackLayout ) Button.Parent;
```

El segundo hijo de esta StackLayout es el ScrollView, entonces el Niños propiedad puede ser indexado para obtener lo siguiente:

```
ScrollView ScrollView = ( ScrollView ) OuterLayout.Children [1];
```

Los Contenido propiedad de este ScrollView es exactamente el StackLayout estabas buscando:

```
StackLayout loggerLayout = ( StackLayout ) ScrollView.Content;
```

Por supuesto, el peligro de hacer algo como esto es que puede cambiar el diseño de algún día y se olvide de cambiar su código-árbol caminar de manera similar. Pero la técnica es muy útil si el código que ensambla su página es independiente de la gestión de eventos desde puntos de vista sobre esa página de códigos.

Compartir pulsaciones de botón

Si un programa contiene múltiples Botón vistas, cada una Botón puede tener su propio hecho clic entrenador de animales. Pero en algunos casos puede ser más conveniente para múltiples Botón vistas al comparten un común hecho clic entrenador de animales.

Considere un programa de la calculadora. Cada uno de los botones etiquetados de 0 a 9, básicamente hace lo mismo

cosa, y que tiene 10 separada hecho clic manejadores de estas 10 botones, incluso si comparten un código común, simplemente no tendría mucho sentido.

Usted ha visto cómo el primer argumento de la hecho clic manejador se puede convertir a un objeto de tipo Botón. Pero, ¿cómo saber cual Botón ¿es?

Un método consiste en almacenar toda la Botón objetos como campos y luego comparar la Botón oponerse a disparar el evento con estos campos.

los **TwoButtons** programa muestra esta técnica. Este programa es similar al programa anterior pero con dos botones, uno para añadir Etiqueta se opone a la StackLayout, y el otro para eliminarlos. Los dos Botón objetos se almacenan como campos de manera que la hecho clic manejador puede determinar cuál generó el evento:

```
clase pública TwoButtonsPage : Pagina de contenido
{
    Botón addButton, RemoveButton;
    StackLayout loggerLayout = nuevo StackLayout();

    público TwoButtonsPage ()
    {
        // Crear el punto de vista de botón y asociar controladores se hace clic.
        addButton = nuevo Botón
        {
            text = "Añadir",
            HorizontalOptions = LayoutOptions .CenterAndExpand
        };
        addButton.Clicked += OnButtonClicked;

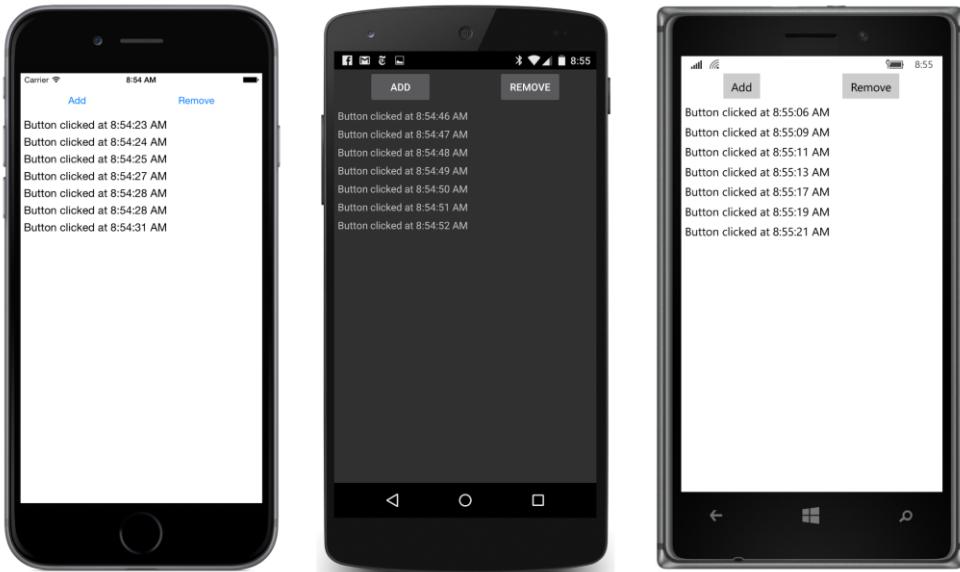
        RemoveButton = nuevo Botón
        {
            text = "Retirar",
            HorizontalOptions = LayoutOptions .CenterAndExpand,
            IsEnabled = falso
        };
        removeButton.Clicked += OnButtonClicked;

        esta .Padding = nuevo Espesor (5, Dispositivo .OnPlatform (20, 0, 0), 5, 0);

        // Montar la página.
        esta .Este contenido = nuevo StackLayout
        {
            Los niños =
            {
                nuevo StackLayout
                {
                    Orientación = StackOrientation .Horizontal,
                    Los niños =
                    {
                        addButton,
                        RemoveButton
                    }
                }
            }
        }
    }
}
```

```
        },  
  
        nuevo ScrollView  
        {  
            VerticalOptions = LayoutOptions.FillAndExpand,  
            Content = loggerLayout  
        }  
    }  
};  
  
vacío OnButtonClicked ( objeto remitente, EventArgs args)  
{  
    Botón botón = ( Botón )remitente;  
  
    Si (Botón == addButton)  
    {  
        // Añadir etiqueta para desplazable StackLayout.  
        loggerLayout.Children.Add ( nuevo Etiqueta  
        {  
            text = "Botón hace clic en" + Fecha y hora .Now.ToString ("T")  
        });  
    }  
    más  
    {  
        // Eliminar la etiqueta superior de StackLayout.  
        loggerLayout.Children.RemoveAt (0);  
    }  
  
    // Habilitar el botón "Eliminar" sólo si los niños están presentes.  
    removeButton.IsEnabled = loggerLayout.Children.Count > 0;  
}  
}
```

Ambos botones se les da una HorizontalOptions valor de CenterAndExpand de modo que se pueden visualizar de lado a lado en la parte superior de la pantalla mediante el uso de una horizontal StackLayout:



Observe que cuando el hecho clic Detecta manejador RemoveButton, simplemente llama al RemoveAt método en el Niños propiedad:

```
loggerLayout.Children.RemoveAt (0);
```

Pero, ¿qué ocurre si no hay niños? no lo hará RemoveAt lanzar una excepción?

No puede suceder! Cuando el **TwoButtons** programa comienza, la Está habilitado propiedad de la retirarBotón se inicializa a falso. Cuando un botón se desactiva de esta manera, un tenues señales de apariencia para el usuario que es no funcional. No proporciona información al usuario y no se dispara hecho clic eventos. Hacia el final de la hecho clic manejador, el Está habilitado propiedad de RemoveButton se establece en cierto sólo si el loggerLayout tiene al menos un hijo.

Esto pone de manifiesto una buena regla general: si el código necesita para determinar si un botón hecho clic evento es válida, es probablemente mejor para evitar clics no válidos botón desactivando el botón.

controladores de eventos anónimos

Al igual que con cualquier controlador de eventos, se puede definir una hecho clic controlador como una función lambda anónima. He aquí un programa llamado **ButtonLambdas** que tiene una Etiqueta mostrando un número y dos botones. Un botón duplica el número, y la otra mitad el número. Normalmente, el número y Etiqueta

las variables se guardan como campos. Pero debido a que los controladores de eventos anónimas se definen derecha en el constructor después se definen estas variables, los controladores de eventos tienen acceso a estas variables locales:

[clase pública ButtonLambdasPage : Pagina de contenido](#)

```
{
```

```
público ButtonLambdasPage ()  
{  
    // Número de manipular.  
    doble número = 1;  
  
    // crear la etiqueta para su visualización.  
    Etiqueta etiqueta = nuevo Etiqueta  
    {  
        Text = Number.toString ().  
        Tamaño de Letra = Dispositivo .GetNamedSize ( NamedSize .Grande, tipo de ( Etiqueta )),  
        HorizontalOptions = LayoutOptions .Centrar,  
        VerticalOptions = LayoutOptions .CenterAndExpand  
    };  
  
    // Crear el primer botón y adjuntar manejador se hace clic.  
    Botón timesButton = nuevo Botón  
    {  
        text = "Doble",  
        Tamaño de Letra = Dispositivo .GetNamedSize ( NamedSize .Grande, tipo de ( Botón )),  
        HorizontalOptions = LayoutOptions .CenterAndExpand  
    };  
    timesButton.Clicked += (remitente, args) =>  
    {  
        número * = 2;  
        label.text = Number.toString ();  
    };  
  
    // Cree el segundo botón y adjuntar manejador se hace clic.  
    Botón divideButton = nuevo Botón  
    {  
        text = "Mitad",  
        Tamaño de Letra = Dispositivo .GetNamedSize ( NamedSize .Grande, tipo de ( Botón )),  
        HorizontalOptions = LayoutOptions .CenterAndExpand  
    };  
    divideButton.Clicked += (remitente, args) =>  
    {  
        número / = 2;  
        label.text = Number.toString ();  
    };  
  
    // Montar la página.  
    esta .Este contenido = nuevo StackLayout  
    {  
        Los niños =  
        {  
            etiqueta,  
            nuevo StackLayout  
            {  
                Orientación = StackOrientation .Horizontal,  
                VerticalOptions = LayoutOptions .CenterAndExpand,  
                Los niños =  
                {  
                    timesButton,  
                    divideButton  
                };  
            };  
        };  
    };  
};
```

Observe el uso de `Device.GetNamedSize` para obtener el texto grande, tanto para el Etiqueta y el Botón. Cuando se utiliza con Etiqueta, el segundo argumento de `GetNamedSize` debe indicar una Etiqueta, y cuando se utiliza con el Botón se debe indicar una Botón. Los tamaños de los dos elementos pueden ser diferentes.

Al igual que el programa anterior, los dos botones comparten una horizontal StackLayout:



La desventaja de definir manejadores de eventos como funciones lambda anónimos es que no se pueden compartir entre varios puntos de vista.
(En realidad se puede, pero algo de código desordenado reflexión está involucrado.)

vistas distintivas con ID

En el **TwoButtons** programa, se vio una técnica para compartir un controlador de eventos que distingue puntos de vista mediante la comparación de objetos. Esto funciona bien cuando no hay muchos puntos de vista para distinguir, pero sería un enfoque terrible para un programa de la calculadora.

Los Elemento clase define una styleid propiedad de tipo cuerda específicamente para el propósito de identificar puntos de vista. No se usa para cualquier cosa interna para Xamarin.Forms, por lo que puede poner lo que es conveniente para la aplicación. Puede probar los valores mediante el uso de Si y más declaraciones o en una cambiar

y caso bloque, o puede utilizar una **Analizar gramaticalmente** Método para convertir las cadenas en números o miembros de la enumeración.

El siguiente programa no es una calculadora, pero es un teclado numérico, que sin duda es parte de una calculadora. El programa se llama **SimplestKeypad** y utiliza una **StackLayout** para la organización de las filas y columnas de teclas. (Uno de los propósitos de este programa es demostrar que **StackLayout** No es bastante la herramienta adecuada para este trabajo!)

El programa crea un total de cinco StackLayout instancias. Los mainStack está orientado verticalmente, y cuatro horizontal StackLayout objetos organizar los botones de 10 dígitos. Para simplificar las cosas, el teclado se arregla con pedido de teléfono en lugar de ordenar la calculadora:

```
público clase SimplestKeypadPage : Pagina de contenido
{
    Etiqueta displayLabel;
    Botón backspaceButton;

    público SimplestKeypadPage ()
    {
        // crear una pila vertical de todo el teclado.
        StackLayout mainStack = nuevo StackLayout
        {
            VerticalOptions = LayoutOptions .Centrar,
            HorizontalOptions = LayoutOptions .Centrar
        };

        // La primera fila es la etiqueta.
        displayLabel = nuevo Etiqueta
        {
            Tamaño de Letra = Dispositivo .GetNamedSize ( NamedSize .Grande, tipo de ( Etiqueta )),
            VerticalOptions = LayoutOptions .Centrar,
            HorizontalTextAlignment = Alineación del texto .Fin
        };
        mainStack.Children.Add (displayLabel);

        // El segunda linea es el botón de retroceso.
        backspaceButton = nuevo Botón
        {
            text = "\u21E6",
            Tamaño de Letra = Dispositivo .GetNamedSize ( NamedSize .Grande, tipo de ( Botón )),
            IsEnabled = falso
        };
        backspaceButton.Clicked += OnBackspaceButtonClicked;
        mainStack.Children.Add (backspaceButton);

        // Ahora haga las teclas 10 numéricas.
        StackLayout rowStack = nulo ;
        para ( En t num = 1; num <= 10; num ++ )
        {
            Si ((Num - 1)% 3 == 0)
            {

```

```

rowStack = nuevo StackLayout
{
    Orientación = StackOrientation.Horizontal
};

mainStack.Children.Add (rowStack);
}

Botón digitButton = nuevo Botón
{
    Text = (% 10 num) .ToString () .
    Tamaño de Letra = Dispositivo .GetNamedSize ( NamedSize .Grande, tipo de ( Botón )),
    StyleId = (num% 10) .ToString ()
};

digitButton.Clicked += OnDigitButtonClicked;

// Para el botón de cero, expandirse para llenar horizontalmente.
Si (== num 10)
{
    digitButton.HorizontalOptions = LayoutOptions .FillAndExpand;
}
rowStack.Children.Add (digitButton);
}

esta .Este contenido ha sido = mainStack;
}

vacío OnDigitButtonClicked ( objeto remitente, EventArgs args)
{
    Botón botón = ( Botón )remitente;
    displayLabel.Text += ( cuerda )Button.StyleId;
    backspaceButton.IsEnabled = cierto ;
}

vacío OnBackspaceButtonClicked ( objeto remitente, EventArgs args)
{
    cuerda text = displayLabel.Text;
    displayLabel.Text = text.Substring (0, text.length - 1);
    backspaceButton.IsEnabled = displayLabel.Text.Length> 0;
}
}

```

Las teclas numéricas 10 comparten un único hecho clic entrenador de animales. los styleid propiedad indica el número asociado con la clave, por lo que el programa puede simplemente añadir el número a la cadena mostrada por la **Etiqueta.** los styleid pasa a ser idéntica a la Texto propiedad de la Botón, y el Texto propiedad podría utilizarse en su lugar, pero en el caso general, las cosas no siempre son bastante que conveniente.

el retroceso Botón es suficientemente diferente de la función para justificar su propia hecho clic manipulador, aunque seguramente sería posible combinar los dos métodos en una sola para tomar ventaja de cualquier código que podría tener en común.

Para dar el teclado de un tamaño un poco más grande, se le da todo el texto de una Tamaño de fuente utilizando NamedSize.Large.

Aquí están las tres versiones del **SimplestKeypad** programa:



Por supuesto, tendrá que pulsar las teclas repetidamente hasta que vea cómo el programa responde a una muy grande cadena de dígitos, y descubrirá que no tiene previsto realizar adecuadamente tal cosa. Cuando el Etiqueta se vuelve demasiado amplia, comienza a regir la anchura total de la vertical, StackLayout, y los botones de desplazamiento empiezan así.

Por otra parte, si los botones contienen letras o símbolos en lugar de números, los botones serán desalineados porque cada ancho de botón se basa en su contenido.

Se puede solucionar este problema con la `expande` bandera en el `HorizontalOptions` ¿propiedad? No. El `expande` indicador hace que el espacio adicional que se distribuirá en partes iguales entre los puntos de vista de la `StackLayout`. Cada vista se incrementará de forma aditiva en la misma cantidad, pero los botones de comenzar con diferentes anchos, y siempre tendrá diferentes anchuras. Por ejemplo, echar un vistazo a los dos botones de la `TwoButtons` o `ButtonLambdas` programa. Esos botones tienen su `HorizontalOptions` propiedades ajustado a `Llenar-AndExpand`, pero son diferentes anchuras porque la anchura del contenido botón es diferente.

Una mejor solución para estos programas es la disposición conocida como la Cuadrícula, que vienen en el capítulo 17.

Guardar los datos transitoria

Supongamos que usted está entrando en un número importante en el `SimplestKeypad` programa y que está interrumpido, tal vez con una llamada telefónica. Más tarde, usted apaga el teléfono, terminando con eficacia el programa.

Lo que debería ocurrir la próxima vez que se ejecuta `SimplestKeypad`? En caso de que la larga cadena de números que ha introducido anteriormente desecharse? O debe parecer como si el programa se reanuda desde el estado

la última vez que lo dejó? Por supuesto, no importa para un programa de demostración simple como **SimplestKeypad**, pero en el caso general, los usuarios esperan que las aplicaciones móviles para recordar exactamente lo que estaban haciendo la última vez que interactuaron con el programa.

Por esta razón, el **Solicitud clase** admite dos instalaciones que ayudan al programa de guardar y restaurar los datos:

- **los propiedades propiedad de Solicitud es un diccionario con cuerda llaves y objeto**
artículos. El contenido de este diccionario se guardan automáticamente antes de la aplicación que se está terminado, y el contenido guardado estén disponibles la próxima vez que se ejecuta la aplicación.
- **los Solicitud clase define tres métodos virtuales protegidos, nombrados OnStart, OnSleep,**
y En resumen, y el Aplicación clase generada por la plantilla Xamarin.Forms anula estos métodos. Estos métodos ayudan a un acuerdo de aplicación con lo que se conoce como *ciclo de vida de aplicaciones*
eventos.

Para utilizar estas instalaciones, es necesario identificar qué información de su aplicación tiene que salvar para que se pueda restaurar su estado después de haber sido terminado y reiniciado. En general, esta es una combinación de *Configuraciones de la aplicación* -como colores y tamaños de fuente que el usuario podría tener la oportunidad de configurar y *datos transitorios*, tales como campos de entrada de medio a entrar. configuración de la aplicación por lo general se aplican a toda la aplicación, mientras que los datos transitoria es único para cada página de la aplicación. Si cada elemento de estos datos es una entrada en el propiedades diccionario, cada elemento necesita una clave de diccionario. Sin embargo, si un programa tiene que guardar un archivo de gran tamaño, como un documento de procesamiento de texto, no debe utilizar el propiedades diccionario, sino que debe tener acceso al sistema de archivos de la plataforma directamente. (Eso es un trabajo para el Capítulo 20, "asíncrono y el archivo de E / S").

También, se debe restringir los tipos de datos utilizados con propiedades a los tipos de datos básicos soportados por .NET y C #, tales como string, int, y doble.

los **SimplestKeypad** programa necesita para salvar a un solo elemento de datos transitorios, y la clave de diccionario "displayLabelText" parece razonable.

A veces, un programa puede utilizar la propiedades diccionario para guardar y recuperar datos sin involucrarse con los eventos del ciclo de vida de aplicaciones. Por ejemplo, el **SimplestKeypad** programa sabe exactamente cuando el Texto propiedad de displayLabel cambios. Sucede sólo en los dos hecho clic controladores de eventos para las teclas numéricas y la tecla de borrar. Esos dos controladores de eventos podrían simplemente almacenar el nuevo valor en el propiedades diccionario.

Pero espera: propiedades es una propiedad de la **Solicitud clase**. ¿Necesitamos a guardar la instancia de la **Aplicación clase** para que en el código de SimplestKeypadPage puede obtener acceso al diccionario? No, no es necesario. **Solicitud** define una propiedad estática llamada **Corriente** que devuelve la instancia de la aplicación actual de la **Solicitud clase**.

Para almacenar el Texto propiedad de la Etiqueta en el diccionario, sólo tiene que añadir la siguiente línea en la parte inferior de los dos hecho clic controladores de eventos en **SimplestKeypad**:

```
Solicitud.Current.Properties [ "DisplayLabelText" ] = DisplayLabel.Text;
```

No se preocupe si el displayLabelText clave aún no existe en el diccionario: La propiedades implementa el diccionario genérico `IDictionary` interfaz, que define explícitamente el indexador para que el punto anterior, si la clave ya existe o para añadir un nuevo elemento al diccionario si no existe la clave. Ese comportamiento es exactamente lo que usted quiere aquí.

los SimplestKeypadPage constructor puede entonces concluir inicializando el Texto propiedad de la Etiqueta con el siguiente código, que recupera el elemento del diccionario:

```
IDictionary < cuerda , objeto > properties = Solicitud .Current.Properties;

Si (Properties.ContainsKey ( "DisplayLabelText" ))
{
    displayLabel.Text = Propiedades [ "DisplayLabelText" ] como cuerda ;
    backspaceButton.IsEnabled = displayLabel.Text.Length> 0;
}
```

Todo esto es tu aplicación tiene que hacer: sólo tienes que guardar la información en el propiedades diccionario y recuperarlo. Xamarin.Forms sí es responsable de la tarea de guardar y cargar el contenido del diccionario de almacenamiento de la aplicación específica de la plataforma.

En general, sin embargo, es mejor para una aplicación para interactuar con el propiedades diccionario de una manera más estructurada, y aquí es donde los eventos del ciclo de vida de aplicaciones entran en juego. Estos son los tres métodos que aparecen en el Aplicación clase generada por la plantilla Xamarin.Forms:

```
público clase Aplicación : Solicitud
{
    público App ()
    {
        ...
    }

    protegido override void OnStart ()
    {
        // manipulador cuando se inicia el app
    }

    protegido override void OnSleep ()
    {
        // manipulador cuando su aplicación duerme
    }

    protegido override void En resumen()
    {
        // manejar cuando sus hojas de vida de aplicaciones
    }
}
```

El más importante es el OnSleep llamada. En general, una aplicación en modo de suspensión cuando ya no manda la pantalla y se ha convertido en inactiva (aparte de algunos trabajos en segundo plano que podría

han iniciado). De este modo el sueño, una aplicación se puede reanudar (señalado por una En resumen llamar) o terminado. Pero esto es importante: **Después de la OnSleep llamar, no hay ninguna otra notificación que se está terminando una aplicación.** Los OnSleep llamada es lo más cerca que se llega a una notificación de terminación, y siempre precede a una terminación. Por ejemplo, si se ejecuta la aplicación y el usuario apaga el teléfono, la aplicación recibe una OnSleep llamar cuando el teléfono se está cerrando.

En realidad, hay algunas excepciones a la regla de que una llamada a OnSleep siempre precede a la finalización del programa: un programa que se estrella no obtiene una OnSleep llame primero, pero es probable que esperas. Pero aquí hay un caso que no podría anticipar: Cuando se depura una aplicación Xamarin.Forms, y el uso de Visual Studio o Xamarin Studio para detener la depuración, el programa se termina sin precedentes OnSleep llamada. Esto significa que cuando se está depurando código que utiliza estos eventos del ciclo de vida de aplicaciones, debe entrar en el hábito de usar el propio teléfono para poner su programa a dormir, para reanudar el programa, y para terminarlo.

Cuando la aplicación se está ejecutando Xamarin.Forms, la forma más fácil de desencadenar una OnSleep llamada en un teléfono o un simulador está presionando el teléfono de **Casa** botón. A continuación, puede poner el programa al primer plano y desencadenar una En resumen llamar seleccionando la aplicación desde el menú de inicio (en los dispositivos iOS o Android) o pulsando el **Espalda** botón (en los dispositivos Android y Windows Phone).

Si el programa se está ejecutando y Xamarin.Forms se invoca comutador de aplicaciones por el teléfono de pulsar el **Casa** botón dos veces en los dispositivos IOS, pulsando el **Tarea múltiple** botón en dispositivos Android (o manteniendo pulsada la **Casa** botón en los dispositivos Android más antiguos), o manteniendo pulsada la **Espalda** botón en un teléfono de Windows-la aplicación obtiene una OnSleep llamada. Si a continuación selecciona ese programa, la aplicación obtiene una En resumen llame a medida que continúa la ejecución. Si en vez termine la aplicación del-deslizando la imagen de la aplicación al alza en los dispositivos iOS o tocando la X en la esquina superior derecha de la imagen de la aplicación en Android y Windows Phone dispositivos el programa deja de ejecutarse sin más notificación.

Así que aquí está la regla básica: Cada vez que su aplicación recibe una llamada a OnSleep, debe asegurarse de que la propiedades El diccionario contiene toda la información acerca de la aplicación que desea guardar.

Si estás usando eventos del ciclo de vida únicamente para guardar y restaurar los datos del programa, no es necesario para manejar la En resumen método. Cuando el programa se hace una En resumen llamada, el sistema operativo ya ha restaurado de forma automática el contenido del programa y el estado. Si desea, puede utilizar En resumen como una oportunidad para limpiar el propiedades diccionario porque tiene la seguridad de conseguir otro OnSleep llamar antes de que termine el programa. Sin embargo, si el programa ha establecido una conexión con una red de servicio o está en proceso de establecer una conexión de tales es posible que desee utilizar En-Currículum para restablecer esa conexión. Tal vez la conexión ha terminado en el intervalo que el programa estuvo inactivo. O tal vez algunos datos nuevos está disponible.

Usted tiene cierta flexibilidad al restaurar los datos de la propiedades diccionario para su aplicación como su programa empieza a correr. Cuando un programa Xamarin.Forms se pone en marcha, la primera oportunidad que tiene que ejecutar algún código en la biblioteca de clases portátil es el constructor de la Aplicación clase. A eso

tiempo, el propiedades diccionario ya ha sido rellenado con los datos guardados en el almacenamiento específico de la plataforma. El siguiente código que se ejecuta generalmente es el constructor de la primera página en la aplicación de la instancia Aplicación constructor. Los OnStart llamar Solicitud (y App) se deduce que, a continuación, un método Overridable llama OnAppearing se llama en la clase de página. Se pueden recuperar los datos en cualquier momento durante este proceso de inicio.

Los datos que una aplicación necesita ahorrar es por lo general en una clase de página, pero el OnSleep override está en el Aplicación clase. Así que de alguna manera la clase de página y la Aplicación clase debe comunicarse. Un método consiste en definir una OnSleep método en la clase de página que guarda los datos en el propiedades diccionario y luego llamar a la página de OnSleep método de la OnSleep método en el App. Este enfoque funciona bien para una sola página de la aplicación; de hecho, la Solicitud clase tiene una propiedad estática llamada Pagina principal que se encuentra en el Aplicación constructor y que la OnSleep método se puede utilizar para obtener acceso a esa página, pero no funciona tan bien para las aplicaciones de varias páginas.

Aquí hay un enfoque algo diferente: En primer lugar, definir todos los datos que necesita para guardar propiedades como públicas en el Aplicación clase, por ejemplo:

```
público clase Aplicación : Solicitud
{
    público App ()
    {
        ...
    }

    público cuerda DisplayLabelText { conjunto ; obtener ; }
    ...
}

}
```

La clase de página (o clases) pueden establecer y recuperar esas propiedades cuando sea conveniente. Los Aplicación clase puede restaurar cualquier tipo de inmuebles de la propiedades diccionario en su constructor antes de crear instancias de la página y puede almacenar las propiedades en el propiedades diccionario en su OnSleep anular.

Ese es el enfoque adoptado por el **PersistentKeypad** proyecto. Este programa es idéntica a **SimplestKeypad** excepto que incluye código para guardar y restaurar el contenido del teclado. Aquí está la Aplicación clase que mantiene una pública DisplayLabelText propiedad que se guarda en el OnSleep anular y cargado en la Aplicación constructor:

```
espacio de nombres PersistentKeypad
{
    clase pública Aplicación : Solicitud
    {
        const string displayLabelText = "DisplayLabelText";

        público App ()
        {
            Si (Properties.ContainsKey (displayLabelText))
            {
                DisplayLabelText = ( cuerda ) Propiedades [displayLabelText];
            }
        }
    }
}
```

Para evitar errores de ortografía, la Aplicación clase define la clave de diccionario cadena como una constante. Es el mismo que el nombre de la propiedad, excepto que comienza con una letra minúscula. Observe que el `DisplayLabelText` propiedad se establece antes de crear instancias `PersistentKeypadPage` por lo que está disponible en el `PersistentKeypadPage` constructor.

Una aplicación con muchos más elementos podría querer consolidarlos en una clase llamada AppSettings (por ejemplo), que serializar la clase a un XML o una cadena JSON y, a continuación, guardar la cadena en el diccionario.

los PersistentKeypadPage clase que accede DisplayLabelText propiedad en su constructor y establece la propiedad en sus dos controladores de eventos:

```
calle pública PersistentKeypadPage : Pagina de contenido
{
    Etiqueta displayLabel;
    Botón backspaceButton;

    público PersistentKeypadPage ()
    {
        ...
    }

    // Nuevo código para cargar texto teclado anterior.
    Aplicación aplicación = Solicitud .Corriente como Aplicación ;
    displayLabel.Text = app.DisplayLabelText;
    backspaceButton.IsEnabled = displayLabel.Text! = nulo &&
        displayLabel.Text.Length > 0;

    }
}
```

```
vacío OnDigitButtonClicked ( objeto remitente, EventArgs args)
{
    Botón botón = ( Botón )remitente;
    displayLabel.Text += ( cuerda ) Button.StyleId;
    backspaceButton.IsEnabled = cierto ;

    // Guardar el texto teclado.
    Aplicación aplicación = Solicitud .Corriente como Aplicación ;
    app.DisplayLabelText = displayLabel.Text;
}

vacío OnBackspaceButtonClicked ( objeto remitente, EventArgs args)
{
    cuerda text = displayLabel.Text;
    displayLabel.Text = text.Substring ( 0, text.length - 1 );
    backspaceButton.IsEnabled = displayLabel.Text.Length> 0;

    // Guardar el texto teclado.
    Aplicación aplicación = Solicitud .Corriente como Aplicación ;
    app.DisplayLabelText = displayLabel.Text;
}
}
```

Al probar programas que utilizan el propiedades eventos del ciclo de vida del diccionario y de aplicación, usted querrá para desinstalar el programa de vez en cuando desde el teléfono o un simulador. Desinstalar un programa desde un dispositivo también se borran todos los datos almacenados, por lo que la próxima vez que el programa se implementa desde Visual Studio o Xamarin Studio, el programa llega a un diccionario vacío, como si se tratara de que se ejecuta por primera vez.

Capítulo 7

código XAML vs.

C # es sin duda uno de los más grandes lenguajes de programación que el mundo haya visto. Puede escribir aplicaciones completas Xamarin.Forms en C #, y es concebible que has encontrado C # para ser tan ideal para Xamarin.Forms que ni siquiera han considerado el uso de cualquier otra cosa.

Pero hay que tener una mente abierta. Xamarin.Forms proporciona una alternativa a C # que tiene algunas ventajas para ciertos aspectos del desarrollo del programa. Esta alternativa es XAML (pronunciado "zammel"), que representa el Lenguaje de Marcado Extensible Application. Al igual que C #, XAML fue desarrollado en Microsoft Corporation, y está a sólo unos pocos años menor que C #.

Como su nombre indica, XAML se adhiere a la sintaxis de XML, el lenguaje de marcado extensible. Este libro se supone que tiene familiaridad con los conceptos básicos y la sintaxis de XML.

En el sentido más general, XAML es un lenguaje de marcado declarativo utilizado para instanciar e inicializar objetos. Esta definición puede parecer demasiado general, y XAML es de hecho bastante flexible. Pero la mayoría de XAML en el mundo real se ha utilizado para la definición visual de usuario interactúa con estructura de árbol característico de los entornos de programación gráfica. La historia de las interfaces de usuario basadas en XAML comienza con la Windows Presentation Foundation (WPF) y continúa con Silverlight, Windows Phone 7 y 8, y Windows 8 y 10. Cada una de estas implementaciones XAML admite un conjunto algo diferente de los elementos visuales definido por la en particular la plataforma. Del mismo modo, la aplicación XAML en Xamarin.Forms soporta los elementos visuales definidas por Xamarin.Forms, tales como Etiqueta, BoxView, Frame, Button,

StackLayout, y Pagina de contenido.

Como hemos visto, una aplicación Xamarin.Forms escrito enteramente en código define generalmente la aparición inicial de su interfaz de usuario en el constructor de una clase que se deriva de Pagina de contenido. Si usted elige utilizar XAML, el margen de beneficio general reemplaza el código constructor. Usted encontrará que XAML proporciona una definición más sucinta y elegante de la interfaz de usuario y tiene una estructura visual que imita mejor la organización del árbol de los elementos visuales de la página.

XAML es también generalmente más fáciles de mantener y modificar de código equivalente. Debido a XAML es XML, sino que también es potencialmente Puede trabajarse con herramienta: XAML puede ser analizada con mayor facilidad y editado por herramientas de software de código C # equivalentes. De hecho, un impulso a principios detrás de XAML era facilitar una colaboración entre programadores y diseñadores: Los diseñadores pueden utilizar herramientas de diseño que generan XAML, mientras que los programadores se centran en el código que interactúa con el marcado. Si bien esta visión ha sido cumplida tal vez sólo rara vez a la perfección, ciertamente sugiere cómo las aplicaciones pueden ser estructurados para dar cabida a XAML. Se utiliza XAML para las imágenes y el código de la lógica subyacente.

Sin embargo, XAML va más allá de simple división del trabajo. Como se verá en un capítulo futuro, es posible definir los enlaces de la derecha en el XAML que vincular los objetos de interfaz de usuario con los datos subyacentes.

Al crear XAML para las plataformas de Microsoft, algunos desarrolladores utilizan herramientas de diseño interactivo como Microsoft Blend, pero muchos otros prefieren escribir a mano XAML. No hay herramientas de diseño están disponibles para Xamarin.Forms, por lo que la escritura es la única opción. Obviamente, todos los ejemplos XAML en este libro son escritas a mano. Pero incluso cuando las herramientas de diseño están disponibles, la capacidad de escribir a mano XAML es una habilidad importante.

La perspectiva de la escritura XAML podría causar cierta consternación entre los desarrolladores por otra razón: XML es notoriamente detallado. Sin embargo, verá que casi inmediatamente XAML es a menudo más concisa que el código C # equivalentes. El poder real de XAML se vuelve evidente sólo de forma incremental, sin embargo, y no será totalmente evidente hasta el capítulo 19, "vistas colección", cuando se utiliza XAML para la construcción de plantillas para los varios elementos que aparecen en una Vista de la lista.

Es natural que los programadores que prefieren lenguajes fuertemente tipados como C # para ser escéptico de un lenguaje de marcas en el que todo es una cadena de texto. Pero veremos en breve cómo XAML es un análogo muy estricto del código de programación de aplicaciones Xamarin.Forms. Por esta razón, es posible que ni siquiera empezar a pensar en XAML como "inflexible" lenguaje de marcas. El analizador XAML hace su trabajo de una manera muy mecánica basado en la infraestructura subyacente API. Uno de los objetivos de este capítulo y el siguiente es desmitificar XAML e iluminar lo que sucede cuando se analiza el XAML.

Sin embargo, el código y marcado son muy diferentes: Código define un proceso de marcado, mientras que define un estado. XAML tiene varias deficiencias que son intrínsecos a lenguajes de marcado: XAML no tiene bucles, sin control de flujo, sin sintaxis cálculo algebraico, y no hay controladores de eventos. Sin embargo, XAML define varias características que ayudan a compensar algunas de estas deficiencias. Vas a ver muchas de estas características en los siguientes capítulos.

Si no desea utilizar XAML, que no es necesario. Todo lo que se puede hacer en XAML se puede hacer en C#. Pero cuidado: A veces los desarrolladores a obtener una pequeña muestra de XAML y se deja llevar y tratar de hacer todo en XAML! Como de costumbre, la mejor regla es "moderación en todas las cosas." Muchas de las mejores técnicas implican la combinación de código y XAML de manera interactiva.

Vamos a empezar esta exploración con algunos fragmentos de código y el código XAML equivalente, y luego ver cómo XAML y código encajan entre sí en una aplicación Xamarin.Forms.

Propiedades y atributos

Aquí está una Xamarin.Forms Etiqueta instancia e inicializado en el código, tanto como podría parecer en el constructor de una clase de página:

```
nuevo Etiqueta
{
    text = "Hola de Código!",
    isVisible = cierto,
    Opacidad = 0,75,
    HorizontalTextAlignment = Alineación del texto .Centrar,
    VerticalOptions = LayoutOptions .CenterAndExpand,
    TextColor = Color .Azul,
```

```

    BackgroundColor = Color.FromRgb (255, 128, 128),
    Tamaño de Letra = Dispositivo.GetNamedSize ( NamedSize.Grande, tipo de ( Etiqueta )), 
    FontAttributes = FontAttributes.bold | FontAttributes.Italic
};


```

Este es un aspecto muy similar Etiqueta ejemplarizado e inicializado en XAML, que se puede ver de inmediato es más concisa que el código equivalente:

```

<Etiqueta Texto = "Hola desde XAML!" 
    Es visible = "Cíerto"
    Opacidad = "0.75"
    HorizontalTextAlignment = "Centrar"
    VerticalOptions = "CenterAndExpand"
    Color de texto = "Azul"
    Color de fondo = "# FF8080"
    Tamaño de fuente = "Grande"
    FontAttributes = "Negrita cursiva" />

```

clases Xamarin.Forms como Etiqueta convertido en elementos XML en XAML. Las propiedades como Texto, Es visible, y atributos XML el resto convertido en XAML.

Deberían ejecutarse en XAML, una clase como Etiqueta debe tener un constructor sin parámetros pública. (En el siguiente capítulo, podrás ver que hay una técnica para pasar argumentos a un constructor en XAML, pero por lo general se utiliza para fines especiales.) Las propiedades establecidas en XAML debe tener pública conjunto descriptores de acceso. Por convención, los espacios que rodean un signo igual en el código pero no en XML (o XAML), pero se puede utilizar la mayor cantidad de espacio en blanco que deseé.

La concisión de la XAML como resultado principalmente de la brevedad de los valores de los atributos, por ejemplo, el uso de la palabra "grande" en lugar de una llamada a la Device.GetNamedSize método. Estas abreviaturas no están integrados en el analizador XAML. El analizador XAML es en cambio asistido por diversas clases de convertidor de definidos específicamente para este propósito.

Cuando el analizador XAML se encuentra con el Etiqueta elemento, se puede utilizar la reflexión para determinar si Xamarin.Forms tiene una clase llamada Etiqueta, y si es así, se puede crear una instancia de esa clase. Ahora ya está listo para inicializar el objeto. Los Texto propiedad es de tipo cuerda, y el valor del atributo es simplemente asignar a esa propiedad.

Debido a XAML es XML, puede incluir caracteres Unicode en el texto utilizando la sintaxis XML estándar. Preceder al valor Unicode decimal con & # (o el valor Unicode hexadecimal con & # X) y seguir con un punto y coma:

```
Texto = "Costo & # X2014; & # X20AC; 123,45"
```

Esos son los valores Unicode para el guión largo y el símbolo euro. Para forzar un salto de línea, utilice el carácter de avance de línea & # x000A, o (porque no se requieren ceros a la izquierda) & # xA, o, en decimal, & # 10.

Los paréntesis angulares, los símbolos de unión, y las comillas tienen un significado especial en XML, por lo que incluyen los caracteres de una cadena de texto, utilice una de las entidades predefinidas estándar:

- & Lt; para <

- > para >
- < para &
- ' para '
- " para "

Las entidades HTML predefinidos como no son compatibles. Para un espacio de no separación utilización en lugar.

Además, en el capítulo 10, "extensiones de marcado XAML", descubrirá que las llaves {} tienen un significado especial en XAML. Si usted necesita para comenzar un valor de atributo con una llave izquierda, comenzará con un par de llaves {} y luego el corchete izquierdo.

Volviendo al ejemplo: El EsVisible y Opacity propiedades de Etiqueta son de tipo bool y duplicó con bool, respectivamente, y estos son tan simple como se podría esperar. El analizador utiliza el XAML Boolean.Parse y Double.Parse métodos para convertir los valores de atributo. Los Boolean.Parse método es sensible a mayúsculas, pero los valores generalmente booleanas se capitalizan como "verdadero" y "falso" en XAML. Los Double.Parse se pasa método una CultureInfo.InvariantCulture argumento, por lo que la conversión no depende de la cultura local del programador o usuario.

Los HorizontalTextAlignment propiedad de Etiqueta es de tipo Alineación del texto, que es una enumeración. Para cualquier propiedad que es un tipo de enumeración, el analizador utiliza el XAML Enum.Parse método de convertir de la cadena de valor.

Los VerticalOptions propiedad es de tipo LayoutOptions, una estructura. Cuando el analizador XAML hace referencia a la LayoutOptions estructura mediante la reflexión, se descubre que la estructura tiene un atributo definido C #:

```
[TypeConverter (tipo de (LayoutOptionsConverter))]  
público struct LayoutOptions  
{  
    ...  
}
```

! (¡Cuidado Esta discusión implica dos tipos de atributos: Atributos tales como XML HorizontalText-Alineación y C # atributos como este TypeConverter).

Los TypeConverter atributo es soportado por una clase llamada TypeConverterAttribute. Este particular TypeConverter en atribuir LayoutOptions hace referencia a una clase llamada LayoutOptionsConverter, que se deriva de una clase abstracta pública llamada TypeConverter que define métodos llamados CanConvertFrom y ConvertFrom. Cuando el analizador XAML se encuentra con este TypeConverter atributo, se crea una instancia del LayoutOptionsConverter. Los VerticalOptions atributo en el XAML se le asigna la cadena "centro", por lo que el analizador XAML pasa esa cadena "centro" a la ConvertFrom método de LayoutOptionsConverter, y fuera hace estallar una LayoutOptions valor. Esto se asigna a la VerticalOptions propiedad de la Etiqueta objeto.

Del mismo modo, cuando el analizador XAML se encuentra con el Color de texto y Color de fondo propiedades, se utiliza la reflexión para determinar que esas propiedades son de tipo Color. Los Color estructura también está adornada con una TypeConverter atributo:

```
[TypeConverter ( tipo de ( ColorTypeConverter ))]
pública struct Color
{
    ...
}
```

Se puede crear una instancia de ColorTypeConverter y experimentar con ella en el código si lo desea. Se acepta definiciones de color en varios formatos: Se puede convertir una cadena como "azul" a la Color azul valor, y el "defecto" y cadenas de "énfasis" a la Color.Default y Color.Accent valores. Color-

TypeConverter también puede analizar cadenas que codifican los valores de rojo-verde-azul, tales como "# FF8080", que es un valor de rojo de 0xFF, un valor verde de 0x80, y un valor de azul también de 0x80.

Todos los valores RGB numéricos comienzan con un prefijo de número-señal, pero que prefijo pueden ser seguidos con ocho, seis, cuatro, o tres dígitos hexadecimales para especificar valores de color con o sin un canal alfa. Aquí está la más amplia sintaxis:

Color de fondo = "#aarrggbbaa"

Cada una de las cartas representa un dígito hexadecimal, en el alfa orden (opacidad), rojo, verde, y azul. Para el canal alfa, tenga en cuenta que 0xFF es completamente opaco y 0x00 es totalmente transparente. Ésta es la sintaxis sin un canal alfa:

Color de fondo = "#rrggbb"

En este caso el valor alfa se establece en 0xFF para opacidad total.

Otros dos formatos le permiten especificar un solo dígito hexadecimal para cada canal:

Color de fondo = "#argb"
Color de fondo = "#RGB"

En estos casos, el dígito se repite para formar el valor. Por ejemplo, CF3 # es el color RGB 0xCC-0xFF0x33. Estos formatos cortos se usan muy poco.

Los Tamaño de fuente propiedad de Etiqueta es de tipo doble. Esto es un poco diferente de las propiedades de tipo LayoutOptions y Color. Los LayoutOptions y Color estructuras son parte de Xamarin.Forms, para que puedan ser marcados con la C # TypeConverter atributo, pero no es posible marcar el .NET

Doble estructura con una TypeConverter atribuir sólo para los tamaños de fuente!

En cambio, el Tamaño de fuente dentro de la propiedad Etiqueta clase tiene la TypeConverter atributo:

```
clase pública Etiqueta : Ver , IFontElement
{
    ...
    [TypeConverter ( tipo de ( FontSizeConverter ))]
    pública doble Tamaño de fuente
    {
        ...
    }
}
```

```

    ...
}

...
}

```

los `FontSizeConverter` clase determina si la cadena que se pasa a que es uno de los miembros de la `NamedSize` enumeración.

Si no, `FontSizeConverter` asume el valor es una doble.

El último conjunto de atributos en el ejemplo es `FontAttributes`. los `FontAttributes` propiedad es una enumeración denominada `FontAttributes`, y usted ya sabe que el analizador XAML maneja tipos de enumeración de forma automática. sin embargo, el `FontAttributes` enumeración tiene un C # banderas atributo de este modo:

```

[ banderas ]
public enum FontAttributes
{
    Ninguno = 0,
    Bold = 1,
    Cursiva = 2
}

```

Por consiguiente, el analizador XAML permite que varios miembros separados por comas:

```
FontAttributes = "Negrita cursiva"
```

Esta demostración de la naturaleza mecánica del analizador XAML debe ser muy buena noticia. Esto significa que usted puede incluir clases personalizadas en XAML, y estas clases puede tener propiedades de los tipos personalizados, o las propiedades pueden ser de tipo estándar, sino permitir que los valores adicionales. Todo lo que necesita es marcar estos tipos o propiedades con un C # `TypeConverter` atribuir y proporcionar una clase que deriva de `TypeConverter`.

sintaxis de la propiedad de elementos

Aquí hay alguna C # que es similar a la `FramedText` código en el Capítulo 4. En un comunicado que crea una instancia

Marco y una Etiqueta y establece el Etiqueta al Contenido propiedad de la Marco:

```

nuevo Marco
{
    OutlineColor = Color .Acento,
    HorizontalOptions = LayoutOptions .Centrar,
    VerticalOptions = LayoutOptions .Centrar,
    content = nuevo Etiqueta
    {
        text = "Saludos, Xamarin.Forms!"
    }
}

```

Pero cuando se empieza a duplicar esta en XAML, que podría llegar a ser un poco frustrado en el punto donde se establece el `Contenido` atributo:

```
<Marco OutlineColor = "Acento"
```

```
HorizontalOptions = "Centrar"
VerticalOptions = "Centrar"
Contenido = "lo que pasa aquí? "/>
```

¿Cómo puede Contenido atributo de ser fijado a un entero Etiqueta ¿objeto?

La solución a este problema es la característica más fundamental de la sintaxis XAML. El primer paso es separar el Marco etiqueta en las etiquetas de inicio y fin:

```
< Marco OutlineColor = "Acento "
    HorizontalOptions = "Centrar "
    VerticalOptions = "Centrar ">

</ Marco >
```

Dentro de esas etiquetas, etiquetas de añadir dos más que consisten en el elemento (Marco) y la propiedad que desea establecer (Contenido) conectado con un período de:

```
< Marco OutlineColor = "Acento "
    HorizontalOptions = "Centrar "
    VerticalOptions = "Centrar ">

< Frame.Content >

</ Frame.Content >
</ Marco >
```

Ahora ponga el Etiqueta dentro de esas etiquetas:

```
< Marco OutlineColor = "Acento "
    HorizontalOptions = "Centrar "
    VerticalOptions = "Centrar ">

< Frame.Content >
    < Etiqueta Texto = "Saludos, Xamarin.Forms! "/>
</ Frame.Content >

</ Marco >
```

Que la sintaxis es la forma de configurar una Etiqueta al Contenido propiedad de la Marco.

Usted podría preguntarse si esta característica XAML viola las reglas de sintaxis XML. No es así. El periodo no tiene ningún significado especial en XML, por lo Frame.Content es una etiqueta XML perfectamente válido. Sin embargo, XAML impone sus propias reglas acerca de las siguientes etiquetas: La Frame.Content las etiquetas deben aparecer dentro de Marco las etiquetas y atributos no se pueden establecer en el Frame.Content etiqueta. El objeto establecido en el Contenido la propiedad aparece como el contenido XML de esas etiquetas.

Una vez que se introduce esta sintaxis, algo de terminología se hace necesaria. En el fragmento de XAML final que se muestra más arriba:

- Marco y Etiqueta son C # objetos expresados como elementos XML. Se les llama *elementos de objeto*.
- OutlineColor, HorizontalOptions, VerticalOptions, y Texto son C # propiedades expresadas como atributos XML. Se les llama *atribuye la propiedad*.

- Frame.Content es una propiedad de C# expresado como un elemento XML, y por lo tanto se le llama *elemento de propiedad*.

Elementos de propiedad son muy comunes en XAML de la vida real. Vas a ver numerosos ejemplos en este capítulo y en los capítulos futuros, y pronto se encontrará elementos de propiedad convertirse en una segunda naturaleza para el uso de XAML. Pero cuidado: A veces los desarrolladores deben recordar tanto que nos olvidamos de lo básico. Incluso después de haber estado usando XAML por un tiempo, es probable que encuentre una situación en la que no parece posible establecer un objeto particular de una propiedad particular. La solución es muy a menudo un elemento de propiedad.

También puede utilizar sintaxis de la propiedad de elementos de propiedades más simples, por ejemplo:

```
<Marco HorizontalOptions = "Centrar" >
    <Frame.VerticalOptions >
        Centrar
    </Frame.VerticalOptions >
    <Frame.OutlineColor >
        Acento
    </Frame.OutlineColor >
    <Frame.Content >
        <Etiqueta >
            <label.text >
                Saludos, Xamarin.Forms!
            </label.text >
        </Etiqueta >
    </Frame.Content >
</Marco >
```

Ahora el VerticalOptions y OutlineColor propiedades de Marco y el Texto propiedad de Etiqueta

Tiene todos los elementos de propiedad convertirse. El valor de estos atributos es siempre el contenido del elemento de propiedad sin comillas.

Por supuesto, no tiene mucho sentido definir estas propiedades como elementos de propiedad. Es innecesariamente prolífico. Pero funciona como debería.

Vayamos un poco más allá: En lugar de establecer HorizontalOptions al "centro" (correspondiente a la propiedad estática LayoutOptions.Center), se puede expresar HorizontalOptions como un elemento de propiedad y establecer en un LayoutOptions valor con sus propiedades individuales conjunto:

```
<Marco >
    <Frame.HorizontalOptions >
        <LayoutOptions Alineación = "Centrar" 
                      expande = "Falso" />
    </Frame.HorizontalOptions >
    <Frame.VerticalOptions >
        Centrar
    </Frame.VerticalOptions >
    <Frame.OutlineColor >
        Acento
    </Frame.OutlineColor >
    <Frame.Content >
```

```

< Etiqueta >
  < label.text >
    Saludos, Xamarin.Forms!
  </ label.text >
</ Etiqueta >
</ Frame.Content >
</ Marco >

```

Y también se puede expresar estas propiedades de LayoutOptions como elementos de propiedad:

```

< Marco >
  < Frame.HorizontalOptions >
    < LayoutOptions >
      < LayoutOptions.Alignment >
        Centrar
      </ LayoutOptions.Alignment >
      < LayoutOptions.Expands >
        Falso
      </ LayoutOptions.Expands >
    </ LayoutOptions >
  </ Frame.HorizontalOptions >
  ...
</ Marco >

```

No se puede establecer la misma propiedad como un atributo de propiedad y un elemento de propiedad. Eso es establecer la propiedad dos veces, y no es permitido. Y recuerda que ninguna otra cosa puede aparecer en las etiquetas propertyelement. El valor que se establece en la propiedad es siempre el contenido XML de esas etiquetas.

Ahora usted debe saber cómo usar una StackLayout en XAML. En primer lugar expresar la Niños propiedad que el elemento de propiedad StackLayout.Children, y luego incluir a los hijos de la StackLayout como contenido XML de las etiquetas de propiedad de elementos. He aquí un ejemplo donde cada niño de la primera StackLayout es otro StackLayout con una orientación horizontal:

```

< StackLayout >
  < StackLayout.Children >
    < StackLayout Orientación = " Horizontal " >
      < StackLayout.Children >
        < BoxView Color = " rojo " />
        < Etiqueta Texto = " rojo " >
          VerticalOptions = " Centrar "
        </ Etiqueta >
      </ StackLayout.Children >
    </ StackLayout >

    < StackLayout Orientación = " Horizontal " >
      < StackLayout.Children >
        < BoxView Color = " Verde " />
        < Etiqueta Texto = " Verde " >
          VerticalOptions = " Centrar "
        </ Etiqueta >
      </ StackLayout.Children >
    </ StackLayout >

    < StackLayout Orientación = " Horizontal " >
      < StackLayout.Children >

```

```
< BoxView Color = "Azul" />
< Etiqueta Texto = "Azul" /
    VerticalOptions = "Centrar" />
</ StackLayout.Children >
</ StackLayout >
</ StackLayout.Children >
</ StackLayout >
```

cada horizontal StackLayout tiene un BoxView con un color y una Etiqueta con ese nombre del color.

Por supuesto, el marcado repetitivo aquí parece bastante miedo! Lo que si se quería mostrar 16 colores? O 140? Es posible tener éxito en un primer momento con un montón de copiar y pegar, pero si se necesita entonces para refinar las imágenes un poco, que estaría en mal estado. En código que haría esto en un bucle, pero XAML no tiene tal característica.

Cuando marcado amenaza con ser demasiado repetitivo, siempre se puede utilizar código. Definición de algunos de una interfaz de usuario en XAML y el resto en código es perfectamente razonable. Pero hay otras soluciones, como se verá en los siguientes capítulos.

Añadir una página XAML a su proyecto

Ahora que usted ha visto algunos fragmentos de XAML, vamos a ver una página entera XAML en el contexto de un programa completo. En primer lugar, crear una solución Xamarin.Forms llamado **CodePlusXaml** utilizando la plantilla solución portátil de biblioteca de clases.

Ahora añadir un XAML Pagina de contenido al PCL. He aquí cómo: En Visual Studio, haga clic en el **CodePlusXaml** proyecto en el **Explorador de la solución**. Seleccionar **Agregar> Nuevo elemento** en el menú. En el **Agregar ítem nuevo** diálogo, seleccione **Visual C# y Cruz-Plataforma** a la izquierda, y **Página forma Xaml** de la lista central. Nombrarlo **CodePlusXamlPage.cs**.

En Xamarin Studio, invoque el menú desplegable en la **CodePlusXaml** proyecto en el **Solución** lista, y seleccione **Agregar> Nuevo archivo**. En el **Archivo nuevo** diálogo, seleccione **formas** a la izquierda y **ContentPage forma Xaml** en la lista central. (Cuidado: También hay una **ContentView forma Xaml** en la lista. ¿Quieres un contenido **página**.) Nombrarlo **CodePlusXamlPage**.

En cualquiera de los casos, se crean dos archivos:

- **CodePlusXamlPage.xaml**, el archivo XAML; y
- **CodePlusXamlPage.xaml.cs**, un archivo de C # (a pesar de la doble extensión extraño en el nombre del archivo).

En la lista de archivos, el segundo archivo se sangra debajo de la primera, lo que indica su estrecha relación. El archivo de C # se refiere a menudo como el **código detrás** del archivo XAML. Contiene código que soporta el marcado. Estos dos archivos ambas contribuyen a una clase llamada **CodePlusXamlPage** que deriva de **Página de contenido**.

Examinemos primero el archivo de código, excluyendo el utilizando directivas, que se ve así:

```
espacio de nombres CodePlusXaml
{
    público clase parcial CodePlusXamlPage : Pagina de contenido
    {
        público CodePlusXamlPage ()
        {
            InitializeComponent ();
        }
    }
}
```

De hecho, es una clase llamada `CodePlusXamlPage` que deriva de `Pagina de contenido`, al igual que lo previsto. Sin embargo, la definición de clase incluye una parcial palabra clave, que por lo general indica que esta es sólo una parte de la `CodePlusXamlPage` definición de clase. En alguna otra parte no debe haber otra definición de clase parcial para `CodePlusXamlPage`. Así que si existe, ¿dónde está? ¡Es un misterio! (Por ahora.)

Otro misterio es la `InitializeComponent` método que llama al constructor. A juzgar únicamente a partir de la sintaxis, parece que este método debe ser definido o heredado por `Pagina de contenido`. Sin embargo, no se encuentra `InitializeComponent` en la documentación de la API.

Vamos a establecer esos dos misterios a un lado temporalmente y mirar el archivo XAML. Las plantillas de Visual Studio y Studio Xamarin generan dos archivos XAML algo diferentes. Si está utilizando Visual Studio, elimine el margen de beneficio para el `Etiqueta` y sustitúirla por `ContentPage.Content` etiquetas de propiedad de elementos para que se vea como la versión en Xamarin Estudio:

```
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " CodePlusXaml.CodePlusXamlPage " >
< ContentPage.Content >
</ ContentPage.Content >
</ Pagina de contenido >
```

El elemento raíz es `Pagina de contenido`, que es la clase que `CodePlusXamlPage` deriva de. Esta etiqueta comienza con dos declaraciones de espacios de nombres XML, los cuales son URI. Sin embargo, no se molestan en comprobar las direcciones web! No hay nada allí. Estos URIs se limitan a indicar que posee el espacio de nombres y cuál es la función que sirve.

El espacio de nombres predeterminado pertenece a Xamarin. Este es el espacio de nombres XML para los elementos del archivo sin prefijo, como el `Pagina de contenido` etiqueta. El URI incluye el año en que este espacio de nombres entró en vigor y la palabra `formas` como una abreviatura de `Xamarin.Forms`.

El segundo espacio de nombres está asociada con un prefijo de `X` por convención, y pertenece a Microsoft. Este espacio de nombres se refiere a los elementos y atributos que son intrínsecos a XAML y se encuentran en cada aplicación XAML. La palabra `WinFX` se refiere a un nombre de una vez utilizado para el .NET Framework 3.0, que introdujo WPF y XAML. El año 2009 se refiere a una especificación XAML en particular, lo que también implica una colección particular de elementos y atributos que se basan en la especificación XAML original, que data de 2006. Sin embargo, `Xamarin.Forms` implementa sólo un subconjunto de los elementos y atributos en el 2009 especificación.

La siguiente línea es uno de los atributos que es intrínseca a XAML, llamado Clase. Porque el X prefijo se usa casi universalmente para este espacio de nombres, este atributo se refiere comúnmente como x: Class y pronunciado "Clase X".

los x: Class atributo puede aparecer sólo en el elemento raíz de un archivo XAML. Se especifica el espacio de nombres de .NET y el nombre de una clase derivada. La clase base de esta clase derivada es el elemento raíz. En otras palabras, este x: Class especificación indica que el CodePlusXamlPage clase en el

CodePlusXaml espacio de nombre se deriva de Pagina de contenido. Esa es exactamente la misma información que la CodePlusXamlPage definición de clase en el archivo CodePlusXamlPage.xaml.cs.

Vamos a añadir algo a este contenido Pagina de contenido en el archivo XAML. Esto requiere el establecimiento de algo a la Contenido propiedad, que en el archivo XAML significa poner algo entre ContentPage.Content. Comience el contenido con una StackLayout, y luego añadir una Etiqueta al Niños propiedad:

```
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " CodePlusXaml.CodePlusXamlPage " >

    < ContentPage.Content >
        < StackLayout >
            < StackLayout.Children >
                < Etiqueta Texto = " Hola desde XAML! " >
                    Es visible = " Ciento "
                    Opacidad = " 0.75 "
                    HorizontalTextAlignment = " Centrar "
                    VerticalOptions = " CenterAndExpand "
                    Color de texto = " Azul "
                    Color de fondo = " # FF8080 "
                    Tamaño de fuente = " Grande "
                    FontAttributes = " Negrita cursiva " />
                </ StackLayout.Children >
            </ StackLayout >
        </ ContentPage.Content >
    </ Pagina de contenido >
```

Ese es el XAML Etiqueta que viste en el comienzo de este capítulo.

Ahora tendrá que cambiar el Aplicación crear una instancia de la clase esta página tal como lo hace con un derivado de sólo código de Pagina de contenido:

```
espacio de nombres CodePlusXaml
{
    clase pública Aplicación : Solicitud
    {
        público App ()
        {
            MainPage = nuevo CodePlusXamlPage ();
        }
        ...
    }
}
```

Ahora puede crear e implementar este programa. Después de hacerlo, es posible aclarar un par de misterios encontradas anteriormente en esta sección:

En Visual Studio, en el **Explorador de la solución**, Selecciona el **CodePlusXaml** proyecto, encontrar el ícono situado en la parte superior con la información sobre herramientas **Mostrar todos los archivos**, y alternar que sucesivamente.

En Xamarin de estudio, en el **Solución** Lista de archivos, invocar el menú desplegable para toda la solución, y seleccione **Opciones de visualización> Mostrar todos los archivos**.

En el **CodePlusXaml** proyecto de biblioteca de clases portátil, encontrar el **obj** carpeta y dentro de éste, el **Depurar** carpeta. Usted verá un archivo llamado **CodePlusXamlPage.xaml.g.cs**. Observe la *gramo* en el nombre de archivo. Que representa *generado*. Aquí está, completa con el comentario que le indica que este archivo es generado por una herramienta:

```
// -----
// <auto-generado>
//     Este código se genera mediante una herramienta.
//     Tiempo de ejecución Versión: 4.0.30319.42000
//
//     Los cambios en este archivo pueden provocar un funcionamiento incorrecto y se perderá si
//     el código se regenera.
// </ auto-generado>
// -----
```

```
espacio de nombres CodePlusXaml {
    utilizando Sistema;
    utilizando Xamarin.Forms;
    utilizando Xamarin.Forms.Xaml;

    público clase parcial CodePlusXamlPage : global :: {Xamarin.Forms.ContentPage

        [System.CodeDom.Compiler.GeneratedCodeAttribute( "Xamarin.Forms.Build.Tasks.XamlG" ,
                                                       "0.0.0.0" )]

        private void InitializeComponent () {
            esta .LoadFromXaml ( tipo de ( CodePlusXamlPage ) );
        }
    }
}
```

Durante el proceso de construcción, el archivo XAML se analiza, y se genera este archivo de código. Tenga en cuenta que se trata de una definición de clase parcial de **CodePlusXamlPage**, que se deriva de **Página de contenido**, y la clase contiene un método llamado **InitializeComponent**.

En otras palabras, es un ajuste perfecto para el archivo de código subyacente **CodePlusXamlPage.xaml.cs**. Una vez generado el archivo **CodePlusXamlPage.xaml.g.cs**, los dos archivos pueden ser compilados juntos como si fueran C # definiciones simplemente normales de clase parcial.

En tiempo de ejecución, la Aplicación una instancia de la clase **CodePlusXamlPage** clase. los **CodePlusXamlPage** constructor (que se define en el archivo de código subyacente) llamadas **InitializeComponent** (se define en el archivo generado), y **InitializeComponent** llamadas **LoadFromXaml**. Este es un método de extensión para Ver se define en la extensión clase en el **Xamarin.Forms.Xaml** montaje. Qué **LoadFromXaml** en realidad depende

si ha elegido para compilar el código XAML o no (como se discute en la siguiente sección). Pero cuando el `InitializeComponent` devuelve el método, toda la página está en su lugar, como si todo lo que había creado una instancia e inicializado en el código en el `CodePlusXamlPage` constructor.

Es posible seguir añadiendo contenido a la página en el constructor del archivo de código subyacente, pero sólo después de que el `InitializeComponent` llamada devuelve. Vamos a aprovechar esta oportunidad para crear otro Etiqueta mediante el uso de un código de inicio de este capítulo:

```
espacio de nombres CodePlusXaml
{
    p\xf3blico clase parcial CodePlusXamlPage : Pagina de contenido
    {
        p\xf3blico CodePlusXamlPage ()
        {
            InitializeComponent ();

            Etiqueta etiqueta = nuevo Etiqueta
            {
                text = "Hola de C\u00f3digo!",
                isVisible = cierto ,
                Opacidad = 0,75,
                HorizontalTextAlignment = Alineaci\u00f3n del texto .Centrar,
                VerticalOptions = LayoutOptions .CenterAndExpand,
                TextColor = Color .Azul,
                BackgroundColor = Color .FromArgb ( 255, 128, 128),
                Tama\u00f1o de Letra = Dispositivo .GetNamedSize ( NamedSize .Grande, tipo de ( Etiqueta )),
                FontAttributes = FontAttributes .bold | FontAttributes .It\u00e1lico
            };

            (Contenido como StackLayout) .Children.Insert (0, etiqueta);
        }
    }
}
```

El constructor concluye accediendo a la `StackLayout` que sabemos que se establece en el Contenido propiedad de la página y la inserción de la Etiqueta en la cima. (En el siguiente capítulo, verá una forma mucho mejor para hacer referencia a objetos en el archivo XAML mediante el uso de la `x: Nombre` atributo.) Puede crear la Etiqueta antes de la `InitializeComponent` llama, pero no se puede agregarlo a la `StackLayout` en ese momento porque `InitializeComponent` es lo que hace que el `StackLayout` (y todos los demás elementos XAML) para ser instanciados. Aquí está el resultado:



Aparte del texto, los dos botones son idénticos.

Usted no tiene que gastar mucho tiempo examinando el archivo de código generado que crea el analizador XAML, pero es útil para entender cómo el archivo XAML juega un papel tanto en el proceso de construcción y durante el tiempo de ejecución. También, a veces un error en el archivo XAML emite una excepción de tiempo de ejecución en el `LoadFromXaml`.

llamas, por lo que es probable que vea el archivo de código generado pop-up con frecuencia, y usted debe saber lo que es.

El compilador XAML

Usted tiene una opción ya sea para compilar el código XAML durante el proceso de construcción. Compilar el código XAML realiza comprobaciones de validez durante el proceso de construcción, se reduce el tamaño del ejecutable, y mejora el tiempo de carga, pero es un poco más nuevo que el enfoque noncompilation, lo que podría haber problemas a veces.

Para indicar que desea compilar todos los archivos XAML en su aplicación, puede insertar el siguiente montaje de atributo en algún lugar de un archivo de código. El lugar más conveniente es presentar los `Assembly.cs` en el **propiedades** carpeta del proyecto PCL:

```
[montaje : XamlCompilation ( XamlCompilationOptions .Compilar)]
```

Se puede poner en otro archivo de C #, sino porque es un atributo de montaje, tiene que estar fuera de cualquier espacio de nombres bloquear. Usted también necesitará una utilizando la directiva de `Xamarin.Forms.Xaml`.

Puede especificar, alternativamente, que el archivo XAML para una clase particular se compila:

```
espacio de nombres CodePlusXaml
{
    [ XamlCompilation ( XamlCompilationOptions .Compilar)]
    público clase parcial CodePlusXamlPage : Pagina de contenido
    {
        público CodePlusXamlPage ()
        {
            InitializeComponent ();

            ...
        }
    }
}
```

los `XamlCompilationOptions` enumeración tiene dos miembros, `Compilar` y `Omitir`, lo que significa que se puede utilizar `XamlCompilation` como un conjunto de atributos para permitir la compilación XAML para todas las clases en el proyecto, pero evita la compilación XAML para las clases individuales mediante el uso de la `Omitir` miembro.

Cuando tu lo hagas *no* optar por compilar el código XAML, todo el archivo XAML está ligada en el ejecutable como un recurso incrustado, al igual que el cuento de Edgar Allan Poe en el **Gato negro** programa en el Capítulo 4. De hecho, usted puede tener acceso al archivo XAML en tiempo de ejecución mediante el uso de la `GetManifestResourceStream`

método. Esto es similar a lo que el `LoadFromXaml` llamar `InitializeComponent` hace. Se carga el archivo XAML y lo analiza por segunda vez, instanciar e inicializar todos los elementos en el archivo XAML a excepción del elemento raíz, que ya existe.

Cuando se elige para compilar el código XAML, este proceso se simplifica un poco, pero el `LoadFromXaml` método todavía tiene que crear una instancia de todos los elementos y construir un árbol visual.

especificidad plataforma en el archivo XAML

Aquí está el archivo XAML para un programa llamado **ScaryColorList** eso es similar a un fragmento de XAML que vimos anteriormente. Pero ahora la repetición es aún más aterradora porque cada elemento de color está rodeado por una Marco:

```
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " ScaryColorList.ScaryColorListPage " >

< ContentPage.Content >
    < StackLayout >
        < StackLayout.Children >
            < Marco OutlineColor = " Acento " >
                < Frame.Content >
                    < StackLayout Orientación = " Horizontal " >
                        < StackLayout.Children >
                            < BoxView Color = " rojo " />
                            < Etiqueta Texto = " rojo " 
                                VerticalOptions = " Centrar " />
                        </ StackLayout.Children >
                    </ StackLayout >
                </ Frame.Content >
            </ Marco >
        </ StackLayout.Children >
    </ StackLayout >
</ ContentPage.Content >
```

```

</ Marco >

< Marco OutlineColor = "Acento" >
  < Frame.Content >
    < StackLayout Orientación = "Horizontal" >
      < StackLayout.Children >
        < BoxView Color = "Verde" />
        < Etiqueta Texto = "Verde" 
          VerticalOptions = "Centrar" />
      </ StackLayout.Children >
    </ StackLayout >
  </ Frame.Content >
</ Marco >

< Marco OutlineColor = "Acento" >
  < Frame.Content >
    < StackLayout Orientación = "Horizontal" >
      < StackLayout.Children >
        < BoxView Color = "Azul" />
        < Etiqueta Texto = "Azul" 
          VerticalOptions = "Centrar" />
      </ StackLayout.Children >
    </ StackLayout >
  </ Frame.Content >
</ Marco >

</ StackLayout.Children >
</ StackLayout >
</ ContentPage.Content >
</ Pagina de contenido >

```

El archivo de código subyacente contiene sólo la llamada norma de InitializeComponent.

Aparte del marcado repetitivo, este programa tiene un problema más práctico: Cuando se ejecuta en iOS, el elemento superior se superpone a la barra de estado. Este problema se puede solucionar con una llamada a Device.OnPlatform en el constructor de la página (tal como se vio en el capítulo 2). Porque Device.OnPlatform establece el Relleno propiedad en la página y no requiere nada en el archivo XAML, que podría ir antes o después de la InitializeComponent llamada.

He aquí una manera de hacerlo:

```

pública clase parcial ScaryColorListPage : Pagina de contenido
{
  pública ScaryColorListPage ()
  {
    Relleno = Dispositivo .OnPlatform ( nuevo Espesor (0, 20, 0, 0),
                                         nuevo Espesor (0),
                                         nuevo Espesor (0));
  }

  InitializeComponent ();
}
}

```

O bien, puede establecer un uniforme Relleno valor para las tres plataformas derecho en el elemento raíz del archivo XAML:

```
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " ScaryColorList.ScaryColorListPage "
    Relleno = " 0, 20, 0, 0 " >
< ContentPage.Content >
    ...
</ ContentPage.Content >
</ Pagina de contenido >
```

Que establece el **Relleno** propiedad de la página. los **ThicknessTypeConverter** clase requiere los valores que se van separados por comas, pero tiene la misma flexibilidad que con el **Espesor** constructor. Puede especificar cuatro valores en el orden izquierda, superior, derecha e inferior; dos valores (el primero de izquierda y derecha, y el segundo para la parte superior e inferior); o un valor.

Sin embargo, también puede especificar los valores específicos de la plataforma justo en el archivo XAML mediante el uso de la **OnPlatform** clase, cuyo nombre sugiere que es similar en función a la **Device.OnPlatform** método estático.

OnPlatform es una clase muy interesante, y vale la pena para obtener una idea de cómo funciona. La clase es genérico, y tiene tres propiedades de tipo T, así como una conversión implícita de sí mismo a T que hace uso de la **Device.OS** valor:

```
público clase OnPlatform <T>
{
    público T {iOS obtener ; conjunto ; }

    público T {Android obtener ; conjunto ; }

    público T {WinPhone obtener ; conjunto ; }

    público estático operador implícita T ( OnPlatform <T> onPlatform)
    {
        // devuelve una de las tres propiedades basado en Device.OS
    }
}
```

En teoría, es posible utilizar el **OnPlatform <T>** clase en el código, tal como esto en el constructor de una Pagina de contenido derivado:

```
Relleno = nuevo OnPlatform < Espesor >
{
    iOS = nuevo Espesor (0, 20, 0, 0),
    Android = nuevo Espesor (0),
    WinPhone = nuevo Espesor (0)
};
```

Se puede establecer una instancia de este **OnPlatform** clase directamente a la **Relleno** debido a que la propiedad **OnPlatform** clase define una conversión implícita de sí mismo a la argumento genérico (en este caso **Espesor**).

Sin embargo, no se debe utilizar OnPlatform en código. Utilizar Device.OnPlatform en lugar. OnPlatform está diseñado para XAML, y la única parte realmente difícil es averiguar cómo especificar el argumento de tipo genérico.

Afortunadamente, la especificación XAML 2009 incluye un atributo diseñado específicamente para clases genéricas, llamado TypeArguments. Debido a que es parte de sí mismo XAML, se utiliza con una X prefijo, por lo que aparece como

x: TypeArguments. Así es cómo OnPlatform se utiliza en XAML para seleccionar entre tres Espesos valores:

```
<OnPlatform x:TypeArguments = "Espesor"
    iOS = "0, 20, 0, 0"
    Androide = "0"
    WinPhone = "0" />
```

En este ejemplo (y en el ejemplo de código anterior), la Androide y WinPhone configuración no es necesaria, ya que son los valores por defecto. Observe que el Espesos cuerdas se pueden fijar directamente a las propiedades porque esas propiedades son de tipo Espesos, y por lo tanto el analizador XAML utilizará el

ThicknessTypeConverter para la conversión de esas cadenas.

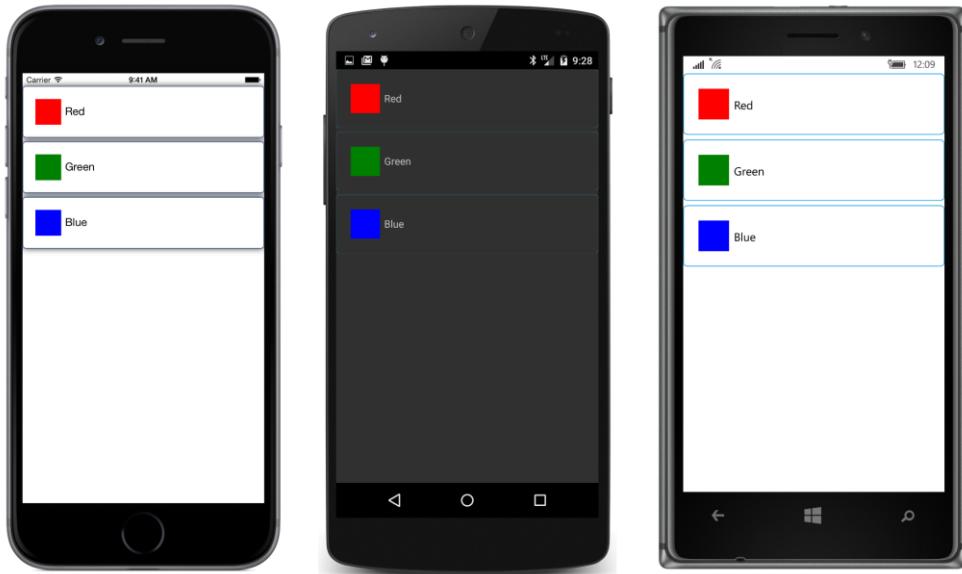
Ahora que tenemos la OnPlatform marcado, ¿cómo lo ponemos a la Relleno propiedad de la Página? expresando Relleno usando sintaxis de la propiedad de elementos, por supuesto!

```
<Pagina de contenido xmlns = "http://xamarin.com/schemas/2014/forms"
    xmlns:x = "http://schemas.microsoft.com/winfx/2009/xaml"
    x:Class = "ScaryColorList.ScaryColorListPage" >

    <ContentPage.Padding >
        <OnPlatform x:TypeArguments = "Espesor"
            iOS = "0, 20, 0, 0" />
    </ContentPage.Padding >

    <ContentPage.Content >
        ...
    </ContentPage.Content >
</Pagina de contenido >
```

Así es como el ScaryColorList programa aparece en la recogida de muestras de este libro y así es como se ve:



Similar a OnDevice, OnIdiom distingue entre Teléfono y Tableta. Por razones que se pondrán de manifiesto en el siguiente capítulo, usted debe tratar de restringir el uso de En el dispositivo y OnIdiom a pequeños trozos de marcadores en vez de grandes bloques. Su uso no debe convertirse en un elemento estructural en sus archivos XAML.

El atributo de propiedad de contenido

El archivo XAML en el **ScaryColorList** programa es en realidad un poco más largo de lo que debe ser. Puede eliminar el ContentPage.Content etiquetas, todo el StackLayout.Children etiquetas, y todo el Frame.Content etiquetas, y el programa funcionarán de la misma:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " ScaryColorList.ScaryColorListPage " >

    < ContentPage.Padding >
        < OnPlatform x: TypeArguments = " Espesor "
            iOS = " 0, 20, 0, 0 " />
    </ ContentPage.Padding >

    < StackLayout >
        < Marco OutlineColor = " Acento " >
            < StackLayout Orientación = " Horizontal " >
                < BoxView Color = " rojo " />
                < Etiqueta Texto = " rojo " /
                    VerticalOptions = " Centrar " />
            </ StackLayout >
        </ Marco >
    </ StackLayout >
```

```

</ Marco >

< Marco OutlineColor = "Acento" >
  < StackLayout Orientación = "Horizontal" >
    < BoxView Color = "Verde" />
    < Etiqueta Texto = "Verde" 
      VerticalOptions = "Centrar" />
  </ StackLayout >
</ Marco >

< Marco OutlineColor = "Acento" >
  < StackLayout Orientación = "Horizontal" >
    < BoxView Color = "Azul" />
    < Etiqueta Texto = "Azul" 
      VerticalOptions = "Centrar" />
  </ StackLayout >
</ Marco >
</ StackLayout >
</ Página de contenido >

```

Se ve mucho más limpio ahora. El elemento única propiedad de la izquierda es para el Relleno propiedad de Página de contenido.

Como con casi todo acerca de la sintaxis XAML, esta eliminación de algunos elementos de la propiedad es apoyado por las clases subyacentes. Cada clase utilizada en XAML se permite definir una propiedad como una *propiedad de contenido* (a veces también denominada *clase de propiedad predeterminada*). Por esta propiedad de contenido, no se requieren las etiquetas de propiedad de elementos, y cualquier contenido XML dentro de las etiquetas de inicio y fin se asignan automáticamente a esta propiedad. Muy convenientemente, la propiedad del contenido Página de contenido es Contenido, la propiedad del contenido StackLayout es Niños, y la propiedad de los contenidos Marco es Contenido.

Estas propiedades de contenido están documentados, pero hay que saber dónde buscar. Una clase especifica su propiedad de contenido mediante el uso de la ContentPropertyAttribute. Si este atributo está unido a una clase, que aparece en la documentación en línea Xamarin.Forms API junto con la declaración de la clase. Así es como aparece en la documentación Página de contenido:

```
[Xamarin.Forms.ContentProperty ("Contenido")]
ContentPage clase pública: TemplatedPage
```

Si usted dice en voz alta, suena un poco redundante: El Contenido propiedad es la propiedad de contenido de Página de contenido.

La declaración de la Marco clase es similar:

```
[Xamarin.Forms.ContentProperty ("Contenido")]
Marco clase pública: ContentView
```

StackLayout no tiene una ContentProperty atribuir aplicado, pero StackLayout deriva de Disposición <Ver>, y Disposición <T> tiene un ContentProperty atributo:

```
[Xamarin.Forms.ContentProperty ("niños")]
Disposición clase abstracta público <T>; Diseño, IViewContainer <T>
donde T: Vista
```

los **ContentProperty** atributo se hereda por las clases que se derivan de Disposición <T>, así que Niños es la propiedad de contenido StackLayout.

Ciertamente, no hay problema si se incluyen los elementos de la propiedad cuando no son necesarios, pero en la mayoría de los casos, ya no aparecerá en los programas de ejemplo en este libro.

El texto con formato

Texto que muestra un archivo XAML puede implicar sólo una o dos palabras, pero a veces se requiere un párrafo entero, tal vez con un poco de formato de carácter integrado. Especificación de formato de carácter no siempre es tan obvio, o tan fácil, en XAML como podría ser sugerido por nuestra familiaridad con el lenguaje HTML.

los **TextVariations** solución tiene un archivo XAML que contiene siete Etiqueta vistas en un desplazable

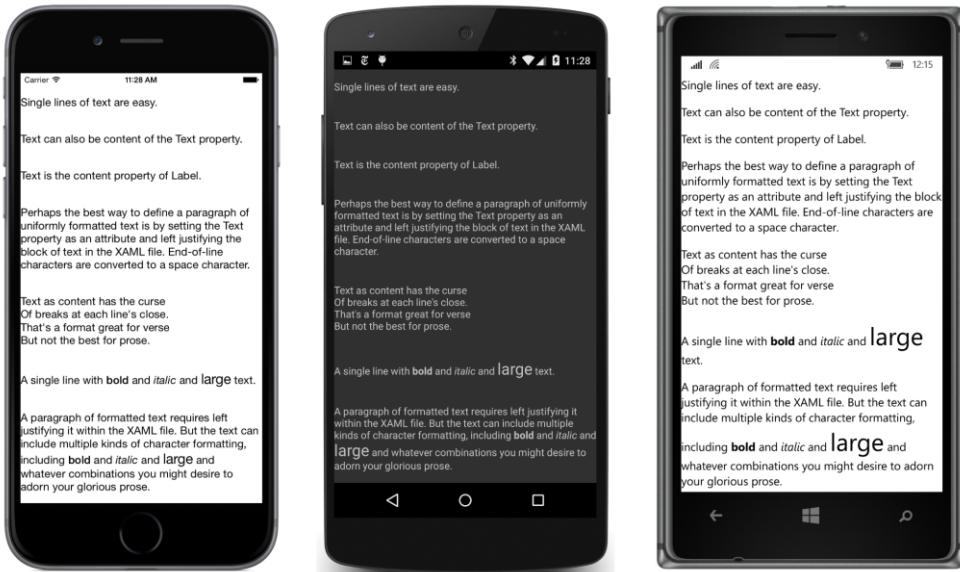
StackLayout:

```
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns:x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " TextVariations.TextVariationsPage " >

    < ContentPage.Padding >
        < OnPlatform x: TypeArguments = " Espesor "
            iOS = " 0, 20, 0, 0 " />
    </ ContentPage.Padding >

    < ScrollView >
        < StackLayout >
            ...
        </ StackLayout >
    </ ScrollView >
</ Pagina de contenido >
```

Cada uno de los siete Etiqueta puntos de vista muestra una manera un tanto diferente de definir el texto que se muestra. Para fines de referencia, aquí está el programa que se ejecuta en las tres plataformas:



El método más sencillo consiste sencillamente en ajustar unas palabras a la Texto atributo de la Etiqueta elemento:

```
<Etiqueta VerticalOptions = "CenterAndExpand">
    Texto = "líneas individuales de texto son fáciles." />
```

También puede establecer la Texto propiedad rompiéndolo como un elemento de propiedad:

```
<Etiqueta VerticalOptions = "CenterAndExpand">
    <label.text>
        El texto también puede ser el contenido de la propiedad Text.
    </label.text>
</Etiqueta>
```

Texto es la propiedad de contenido Etiqueta, por lo que no es necesario el label.text tags:

```
<Etiqueta VerticalOptions = "CenterAndExpand">
    El texto es la propiedad de contenido de la etiqueta.
</Etiqueta>
```

Al configurar el texto como el contenido de la etiqueta (si se utiliza el label.text etiquetas o no), el texto se recorta: todo el espacio blanco, incluidos los retornos de carro, se retira desde el principio y al final del texto. Sin embargo, todo el espacio blanco incrustado se retiene, incluyendo caracteres de fin de línea.

Cuando se establece el Texto propiedad como un atributo de propiedad, todo el espacio en blanco dentro de las comillas se retiene, pero si el texto ocupa más de una línea en el archivo XAML, cada carácter de fin de línea (o secuencia de caracteres) se convierte en un solo espacio.

Como resultado, mostrando un párrafo entero de texto con formato uniforme es algo problemático. El método más infalible es la creación Texto como un atributo de propiedad. Usted puede poner el párrafo entero como una

una sola línea en el archivo XAML, pero si usted prefiere usar múltiples líneas, debe justificación a la izquierda del párrafo entero en el archivo XAML entre comillas, así:

```
<Etiqueta VerticalOptions = "CenterAndExpand">  
    Texto =  
        " Tal vez la mejor manera de definir un párrafo de  
        uniformemente texto formateado está estableciendo el texto  
        propiedad como un atributo y justificación izquierda  
        el bloque de texto en el archivo XAML. Fin de la línea  
        caracteres se convierten en un carácter de espacio. "/>
```

Los caracteres de fin de línea se convierten a caracteres de espacio para que las líneas individuales se concaténan correctamente. Pero ten cuidado: no deje ningún carácter callejeros al final o al principio de las líneas individuales. Los que se mostrará como caracteres extraños dentro del párrafo.

Cuando varias líneas de texto se especifican como contenido de la Etiqueta, sólo espacio en blanco al principio y al final del texto se recorta. Todo el espacio blanco incrustado se retiene, incluyendo caracteres de fin de línea:

```
<Etiqueta VerticalOptions = "CenterAndExpand">  
    El texto tiene como contenido la maldición  
    De las interrupciones en el cierre de cada línea.  
    Eso es un formato grande para el verso  
    Pero no es el mejor para la prosa.  
</Etiqueta>
```

Este texto se representa como cuatro líneas separadas. Si estás mostrando listas o la poesía en su aplicación Xamarin.Forms, eso es exactamente lo que quiere. De lo contrario, probablemente no.

Si su línea o párrafo de texto requiere algún formato de párrafo no uniforme, tendrá que utilizar el FormattedText propiedad de Etiqueta. Como se puede recordar, lo ajusta a una FormattedString objeto y múltiple a continuación, establecer Lapso se opone a la palmos colección de la FormattedString. En XAML, necesita etiquetas de propiedad de elementos para Label.FormattedString, pero palmos es la propiedad de contenido

FormattedString:

```
<Etiqueta VerticalOptions = "CenterAndExpand">  
    <Label.FormattedText>  
        <FormattedString>  
            <Lapso Texto = " Una sola línea con ">  
                <Lapso Texto = " negrita " FontAttributes = " Negrita " />  
                <Lapso Texto = " y ">  
                <Lapso Texto = " itálico " FontAttributes = " Itálico " />  
                <Lapso Texto = " y ">  
                <Lapso Texto = " grande " Tamaño de fuente = " Grande " />  
                <Lapso Texto = " texto. ">  
            </FormattedString>  
        </Label.FormattedText>  
</Etiqueta>
```

Observe que el Texto propiedades de los elementos nonformatted tienen espacios al final o al principio de la cadena de texto, o ambos, de modo que los artículos no se ejecutan en la otra.

En el caso general, sin embargo, usted podría estar trabajando con un párrafo entero. Puede establecer la `Texto` atributo de la `Lapso` a una larga lista, o se puede envolver en varias líneas. Al igual que con `Etiqueta`, mantener a todo el bloque justificado a la izquierda en el archivo XAML:

```
< Etiqueta VerticalOptions = "CenterAndExpand " >
    < Label.FormattedText >
        < FormattedString >
            < Lapso Texto =
                " Un párrafo de texto con formato requiere justificación izquierda
                dentro del archivo XAML. Pero el texto puede incluir múltiples
                tipo de formato de caracteres, incluidos " />
                < Lapso Texto = " negrita " FontAttributes = " Negrita " />
                < Lapso Texto = " y " />
                < Lapso Texto = " itálico " FontAttributes = " Itálico " />
                < Lapso Texto = " y " />
                < Lapso Texto = " grande " Tamaño de fuente = " Grande " />
                < Lapso Texto =
                    " y todo lo que pueda desear combinaciones para adornar
                    su gloriosa prosa. " />
            </ FormattedString >
        </ Label.FormattedText >
    </ Etiqueta >
```

Se dará cuenta de que en la pantalla el texto con el tamaño de fuente grande está alineado con el texto normal en la línea de base, que es el enfoque adecuado tipográficamente, y el espaciado de línea se ajusta para acomodar el texto más grande.

En la mayoría de los programas de Xamarin.Forms, ni tampoco XAML código existen en forma aislada, sino trabajar juntos. Los elementos en XAML pueden desencadenar acontecimientos manejados en el código, y el código puede modificar elementos en XAML. En el siguiente capítulo se verá cómo funciona esto.

Capítulo 8

Código y XAML en armonía

Un archivo de código y un archivo XAML siempre existen como un par. Los dos archivos se complementan entre sí. A pesar de ser conocido como el archivo "código subyacente" para el XAML, muy a menudo el código es prominente en la adopción de las partes más activas e interactivas de la aplicación. Esto implica que el archivo de código subyacente debe ser capaz de hacer referencia a elementos definidos en XAML con tanta facilidad como los objetos instanciados en el código. Del mismo modo, los elementos de XAML deben ser capaces de disparar eventos que se manejan en los controladores de eventos basados en códigos. Eso es lo que este capítulo se trata.

Pero primero, vamos a explorar un par de técnicas inusuales para crear instancias de objetos de un archivo XAML.

pasar argumentos

Cuando se ejecuta una aplicación que contiene un archivo XAML, cada elemento en el archivo XAML se crea una instancia con una llamada al constructor sin parámetros de la clase o estructura correspondiente. El proceso de carga continúa con la inicialización del objeto resultante estableciendo las propiedades de valores de atributo. Esto parece razonable. Sin embargo, los desarrolladores que utilizan XAML tienen a veces una necesidad de crear instancias de objetos con los constructores que requieren argumentos o llamando a un método de creación estática. Estas necesidades generalmente no implican la API en sí, sino que implican clases de datos externos referenciados por el archivo XAML que interactuar con la API.

La especificación XAML 2009 introdujo una `x:Arguments` elemento y una `x:FactoryMethod` atributo para estos casos, y Xamarin.Forms los soporta. Estas técnicas no se utilizan a menudo en circunstancias normales, pero se debe ver cómo funcionan en caso de que surja la necesidad.

Constructores con argumentos

Para pasar argumentos a un constructor de un elemento en XAML, el elemento debe ser separado en las etiquetas de inicio y fin.

Siga la etiqueta de inicio del elemento con `x:Arguments` de inicio y fin. dentro de los `x:Arguments` etiquetas, incluyen uno o más argumentos de constructor.

Pero, ¿cómo se especifican varios tipos de argumentos comunes, tales como doble o `En t?` ¿Se separan los argumentos con comas?

No. Cada argumento debe ser delimitado con las etiquetas de inicio y fin. Afortunadamente, la especificación XAML 2009 define elementos XML para tipos básicos comunes. Puede utilizar estas etiquetas para aclarar los tipos de elementos, para especificar los tipos genéricos de OnPlatform, o delimitar el argumento de su constructor. Aquí está el conjunto completo apoyo de Xamarin.Forms. Observe que duplican los nombres de tipos de .NET en lugar de los nombres de tipo C #:

- x: Objeto
- x: Boolean
- x: Byte
- x: Int16
- x: Int32
- x: Int64
- x: Individual
- x: Doble
- x: Decimal
- x: Char
- x: String
- x: TimeSpan
- x: Array
- x: DateTime (apoyada por Xamarin.Forms pero no la especificación XAML 2009)

Usted encontrará que es difícil encontrar e apuros para encontrar un uso para todos ellos, pero que sin duda va a descubrir usos para algunos de ellos.

los ParameteredConstructorDemo ejemplo muestra el uso de x: Argumentos con argumentos delimitados por x: Doble etiquetas usando tres constructores diferentes de la Color estructura. El constructor con tres parámetros requiere rojo, verde, y azul que van de 0 a 1. El constructor con cuatro parámetros añade un canal alfa como el cuarto parámetro (que se establece aquí a 0.5), y el constructor con un solo parámetro indica una tono de gris de 0 (negro) a 1 (blanco):

```
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " ParameteredConstructorDemo.ParameteredConstructorDemoPage ">

    < StackLayout >
        < BoxView WidthRequest = " 100 "
            HeightRequest = " 100 "
            HorizontalOptions = " Centrar "
            VerticalOptions = " CenterAndExpand " >
            < BoxView.Color >
                < Color >
                    < x: Argumentos >
                        < x: Doble > 1 </ x: Doble >
                        < x: Doble > 0 </ x: Doble >
                        < x: Doble > 0 </ x: Doble >
                    </ x: Argumentos >
                </ Color >
            </ BoxView.Color >
        </ BoxView >
    </ StackLayout >
</ Pagina de contenido >
```

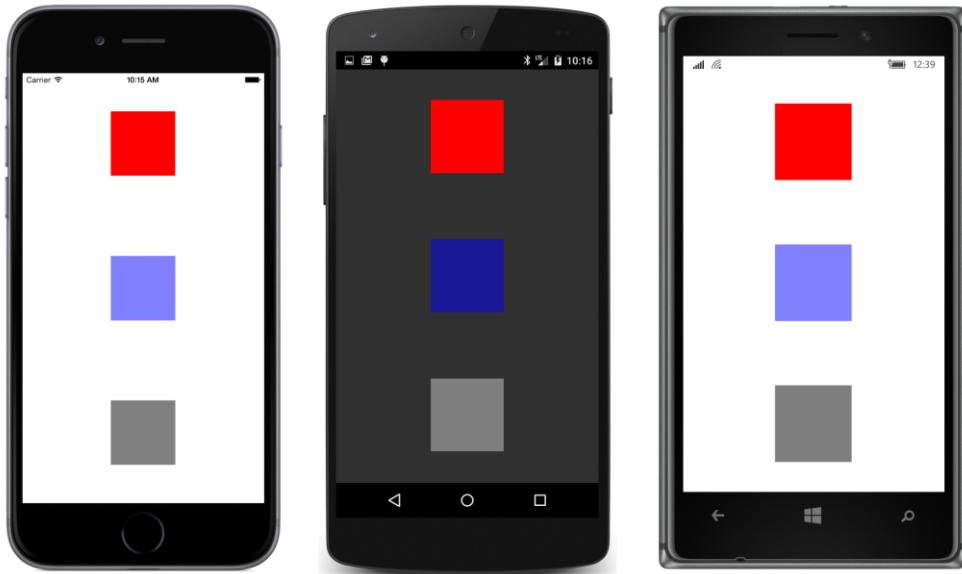
```
</ BoxView.Color >
</ BoxView >

< BoxView WidthRequest = " 100 "
    HeightRequest = " 100 "
    HorizontalOptions = " Centrar "
    VerticalOptions = " CenterAndExpand " >
    < BoxView.Color >
        < Color >
            < x: Argumentos >
                < x: Doble > 0 </ x: Doble >
                < x: Doble > 0 </ x: Doble >
                < x: Doble > 1 </ x: Doble >
                < x: Doble > 0.5 </ x: Doble >
            </ x: Argumentos >
        </ Color >
    </ BoxView.Color >
</ BoxView >

< BoxView WidthRequest = " 100 "
    HeightRequest = " 100 "
    HorizontalOptions = " Centrar "
    VerticalOptions = " CenterAndExpand " >
    < BoxView.Color >
        < Color >
            < x: Argumentos >
                < x: Doble > 0.5 </ x: Doble >
            </ x: Argumentos >
        </ Color >
    </ BoxView.Color >
</ BoxView >

</ StackLayout >
</ Pagina de contenido >
```

El número de elementos dentro de la `x: Argumentos` etiquetas, y los tipos de estos elementos, deben coincidir con uno de los constructores de la clase o estructura. Aquí está el resultado:



El azul BoxView es la luz contra el fondo claro y oscuro sobre el fondo oscuro porque es un 50 por ciento transparente y permite que el programa de fondo a través.

¿Puedo llamar a los métodos de XAML?

En un momento, la respuesta a esta pregunta es "No seas ridícula," pero ahora es un "sí." No te emociones demasiado, sin embargo. Los únicos métodos que puede llamar en XAML son aquellos que devuelven objetos (o valores) del mismo tipo como la clase (o estructura) que define el método. Estos métodos deben ser públicas y estático. A veces se llaman *métodos de creación* o *métodos de fábrica*. Puede crear una instancia de un elemento en XAML a través de una llamada a uno de estos métodos, especificando el nombre del método que utiliza la

x: FactoryMethod atributo y sus argumentos utilizando el x: Argumentos elemento.

los Color estructura define siete métodos estáticos que devuelven Color valores, por lo que estos reúnen los requisitos. Este archivo XAML hace uso de tres de ellos:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
      xmlns:x = " http://schemas.microsoft.com/winfx/2009/xaml "
      x: Class = " FactoryMethodDemo.FactoryMethodDemoPage " >

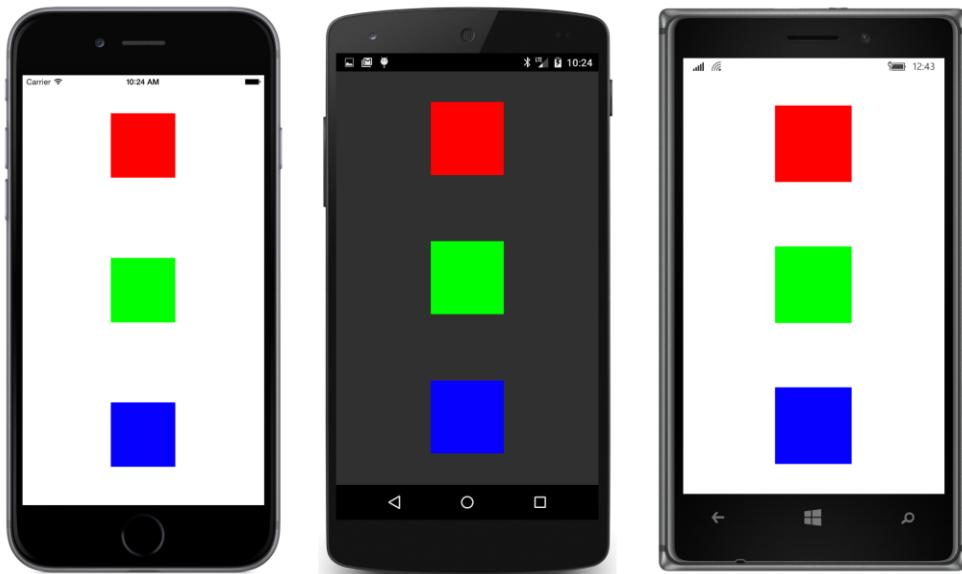
    < StackLayout >
        < BoxView WidthRequest = " 100 "
                  HeightRequest = " 100 "
                  HorizontalOptions = " Centrar "
                  VerticalOptions = " CenterAndExpand " >
            < BoxView.Color >
                < Color x: FactoryMethod = " FromRgb " >
                    < x: Argumentos >
                        < x: Int32 > 255 </ x: Int32 >
                        < x: Int32 > 0 </ x: Int32 >
                    </ x: Argumentos >
                </ Color >
            </ BoxView.Color >
        </ BoxView >
    </ StackLayout >
</ Página de contenido >
```

```
<x:Int32> 0 </x:Int32>
</x:Argumentos>
</Color>
<BoxView.Color>
</BoxView>

<BoxView WidthRequest = "100"
HeightRequest = "100"
HorizontalOptions = "Centrar"
VerticalOptions = "CenterAndExpand">
<BoxView.Color>
<Color x:FactoryMethod = "FromRgb">
<x:Argumentos>
<x:Doble> 0 </x:Doble>
<x:Doble> 1.0 </x:Doble>
<x:Doble> 0 </x:Doble>
</x:Argumentos>
</Color>
</BoxView.Color>
</BoxView>

<BoxView WidthRequest = "100"
HeightRequest = "100"
HorizontalOptions = "Centrar"
VerticalOptions = "CenterAndExpand">
<BoxView.Color>
<Color x:FactoryMethod = "FromHsla">
<x:Argumentos>
<x:Doble> 0.67 </x:Doble>
<x:Doble> 1.0 </x:Doble>
<x:Doble> 0.5 </x:Doble>
<x:Doble> 1.0 </x:Doble>
</x:Argumentos>
</Color>
</BoxView.Color>
</BoxView>
</StackLayout>
</Pagina de contenido>
```

Los dos primeros métodos estáticos invocados en este caso son nombrados Color.FromRgb, pero los tipos de elementos dentro de la x: Argumentos etiquetas distinguen entre Ent argumentos que van de 0 a 255 y doble argumentos que van desde 0 a 1. La tercera es la Color.FromHsla método, lo que crea una Color valor a partir de componentes de matiz, saturación, luminosidad, y alfa. Curiosamente, esta es la única manera de definir una Color valor de los valores HSL en un archivo XAML mediante el uso de la API Xamarin.Forms. Aquí está el resultado:



El atributo x: Nombre

En la mayoría de las aplicaciones reales, el archivo de código subyacente necesita para hacer referencia a elementos definidos en el archivo XAML. Que viste una manera de hacer esto en el [CodePlusXaml](#) programa en el capítulo anterior: si el archivo de código subyacente tiene conocimiento de la disposición del árbol visual se define en el archivo XAML, puede empezar desde el elemento raíz (la propia página) y localizar elementos específicos dentro del árbol. Este proceso se llama "recorrer el árbol" y puede ser útil para la localización de determinados elementos en una página.

En general, un mejor enfoque es dar elementos en el archivo XAML un nombre similar a un nombre de variable. Para ello se utiliza un atributo que es intrínseca a XAML, llamado **Nombre**. Debido a que el prefijo X se usa casi universalmente para los atributos intrínsecos a XAML, este **Nombre** atributo se refiere comúnmente como **x: Nombre**.

los **XamlClock** proyecto demuestra el uso de **x: Nombre**. Aquí está el archivo que contiene dos **XamlClockPage.xaml** Etiqueta controles, nombrados **EtiquetaTiempo** y **dateLabel**:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " XamlClock.XamlClockPage " >

< StackLayout >
    < Etiqueta x: Nombre = " EtiquetaTiempo " 
        Tamaño de fuente = " Grande "
        HorizontalOptions = " Centrar "
        VerticalOptions = " EndAndExpand " />

    < Etiqueta x: Nombre = " dateLabel "
        HorizontalOptions = " Centrar "
        VerticalOptions = " StartAndExpand " />
```

```
</ StackLayout >  
</ Pagina de contenido >
```

Las reglas para x: Nombre son los mismos que para los nombres de variables # C. (Usted verá por qué en breve.) El nombre debe comenzar con una letra o un guión bajo y sólo puede contener letras, guiones y números.

Al igual que el programa de reloj en el capítulo 5, **XamlClock** usos Device.StartTimer para disparar un evento periódico de actualización de la hora y la fecha. Aquí está la XamlClockPage archivo de código subyacente:

```
espacio de nombres XamlClock  
{  
    público clase parcial XamlClockPage  
    {  
        público XamlClockPage ()  
        {  
            InitializeComponent ();  
  
            Dispositivo .StartTimer ( Espacio de tiempo .FromSeconds (1), OnTimerTick);  
        }  
  
        bool OnTimerTick ()  
        {  
            Fecha y hora dt = Fecha y hora .Ahora;  
            timeLabel.Text = dt.ToString ( "T" );  
            dateLabel.Text = dt.ToString ( "RE" );  
            return true ;  
        }  
    }  
}
```

Este método de devolución de llamada de temporizador se llama una vez por segundo. El método debe devolver cierto para continuar con el temporizador. Si devuelve falso, el temporizador se detiene y debe reiniciarse con otra llamada a Device.Start.

Minutero.

Las referencias de métodos de devolución de llamada EtiquetaTiempo y dateLabel como si fueran variables normales y establece el Texto propiedades de cada:



Esto no es un reloj visualmente impresionante, pero definitivamente es funcional.

¿Cómo es que el archivo de código subyacente puede hacer referencia a los elementos identificados con `x: Nombre`? ¿Es magia? Por supuesto no. El mecanismo es muy evidente cuando se examina el archivo de `XamlClockPage.xaml.g.cs` que el analizador XAML genera a partir del archivo XAML medida que el proyecto se está construyendo:

```
// -----
// <auto-generado>
//     Este código se genera mediante una herramienta.
//     Tiempo de ejecución Versión: 4.0.30319.42000
//
//     Los cambios en este archivo pueden provocar un funcionamiento incorrecto y se perderá si
//     el código se regenera.
// </ auto-generado>
// -----
```

```
espacio de nombres XamlClock {
    utilizando Sistema;
    utilizando Xamarin.Forms;
    utilizando Xamarin.Forms.Xaml;

pública clase parcial XamlClockPage : global :: Xamarin.Forms.ContentPage

    [System.CodeDom.Compiler.GeneratedCodeAttribute ("Xamarin.Forms.Build.Tasks.XamlG",
        "0.0.0")]
    privada global :: EtiquetaTiempo Xamarin.Forms.Label;

    [System.CodeDom.Compiler.GeneratedCodeAttribute ("Xamarin.Forms.Build.Tasks.XamlG",
        "0.0.0")]
    privada global :: dateLabel Xamarin.Forms.Label;
```

```
[System.CodeDom.Compiler.GeneratedCodeAttribute ("Xamarin.Forms.Build.Tasks.XamlG",
    "0.0.0.0")]
private void InitializeComponent () {
    esta .LoadFromXaml ( tipo de ( XamlClockPage ));
    EtiquetaTiempo = esta .FindByName < global :: Xamarin.Forms.Label > ("EtiquetaTiempo");
    dateLabel = esta .FindByName < global :: Xamarin.Forms.Label > ("DateLabel");
}
}
```

Puede ser que sea un poco difícil de ver debido a los atributos y tipos completos, pero a medida que el tiempo de acumulación mastica XAML analizador a través del archivo XAML, cada `x: Nombre` atributo se convierte en un campo privado en este archivo de código generado. Esto permite que el código en el archivo de código subyacente para hacer referencia a estos nombres como si fueran campos -que normales que son definitivamente. Sin embargo, los campos son inicialmente nulo. Sólo cuando `InitializeComponent` se llama en tiempo de ejecución son los dos campos establecidos a través de la `FindByName` método, que se define en el `NameScopeExtensions` clase. Si el constructor de su archivo de código subyacente intenta hacer referencia a estos dos campos anteriores a la `InitializeComponent` llamar, tendrán nulo valores.

Este archivo de código generado también implica otra regla para `x: Nombre` valores que ahora es muy obvio, pero rara vez se indique explícitamente: los nombres no pueden duplicar nombres de campos o propiedades definidas en el archivo de código subyacente.

Debido a que estos son campos privados, que sólo se puede acceder desde el archivo de código subyacente y no de otras clases. Si una Pagina de contenido necesidades derivadas exponen a campos públicos o propiedades a otras clases, debe definir los mismo.

Obviamente, `x: Nombre` Los valores deben ser únicos dentro de una página XAML. Esto a veces puede ser un problema si está utilizando `OnPlatform` para los elementos específicos de la plataforma en el archivo XAML. Por ejemplo, aquí hay un archivo XAML que expresa la iOS, Android, y WinPhone propiedades de `OnPlatform` como elementos de propiedades para seleccionar uno de los tres Etiqueta puntos de vista:

```
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns:x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " PlatformSpecificLabels.PlatformSpecificLabelsPage " >

    < OnPlatform x: TypeArguments = " Ver " >
        < OnPlatform.iOS >
            < Etiqueta Texto = " Este es un dispositivo iOS "
                HorizontalOptions = " Centrar "
                VerticalOptions = " Centrar " />
        </ OnPlatform.iOS >

        < OnPlatform.Android >
            < Etiqueta Texto = " Se trata de un dispositivo Android "
                HorizontalOptions = " Centrar "
                VerticalOptions = " Centrar " />
        </ OnPlatform.Android >

        < OnPlatform.WinPhone >
            < Etiqueta Texto = " Se trata de un dispositivo con Windows "
                HorizontalOptions = " Centrar "
                VerticalOptions = " Centrar " />
        </ OnPlatform.WinPhone >
    </ OnPlatform >

```

```

    HorizontalOptions = "Centrar"
    VerticalOptions = "Centrar" />
</OnPlatform.WinPhone>
</OnPlatform>
</Pagina de contenido>

```

los x: TypeArguments atributo de la OnPlatform debe coincidir con el tipo de la propiedad de destino exactamente. Esta OnPlatform elemento de forma implícita se establece en el Contenido propiedad de Pagina de contenido, y esto

Contenido propiedad es de tipo Ver, entonces el x: TypeArguments atributo de la OnPlatform debe especificar

Ver. Sin embargo, las propiedades de OnPlatform se puede ajustar a cualquier clase que deriva de ese tipo. Los objetos puestos a la iOS, Android, y WinPhone propiedades pueden de hecho ser diferentes tipos con tal de que todos ellos se derivan de Ver.

A pesar de que el archivo XAML funciona, no es exactamente óptima. Los tres Etiqueta vistas se crean instancias e inicializados, pero sólo uno se establece en el Contenido propiedad de la Pagina de contenido. El problema con este enfoque surge si necesita hacer referencia a la Etiqueta desde el archivo de código subyacente y darle a cada uno de ellos el mismo nombre, así:

El archivo XAML siguiente no funciona!

```

<Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns:x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x:Class = "PlatformSpecificLabels.PlatformSpecificLabelsPage ">

    <OnPlatform x: TypeArguments = "Ver">
        <OnPlatform.iOS>
            <Etiqueta x: Nombre = "DEVICELABEL"-
                Texto = "Este es un dispositivo iOS"
                HorizontalOptions = "Centrar"
                VerticalOptions = "Centrar" />
        </OnPlatform.iOS>

        <OnPlatform.Android>
            <Etiqueta x: Nombre = "DEVICELABEL"-
                Texto = "Se trata de un dispositivo Android"
                HorizontalOptions = "Centrar"
                VerticalOptions = "Centrar" />
        </OnPlatform.Android>

        <OnPlatform.WinPhone>
            <Etiqueta x: Nombre = "DEVICELABEL"-
                Texto = "Se trata de un dispositivo con Windows"
                HorizontalOptions = "Centrar"
                VerticalOptions = "Centrar" />
        </OnPlatform.WinPhone>
    </OnPlatform>
</Pagina de contenido>

```

Esto no va a funcionar porque varios elementos no pueden tener el mismo nombre.

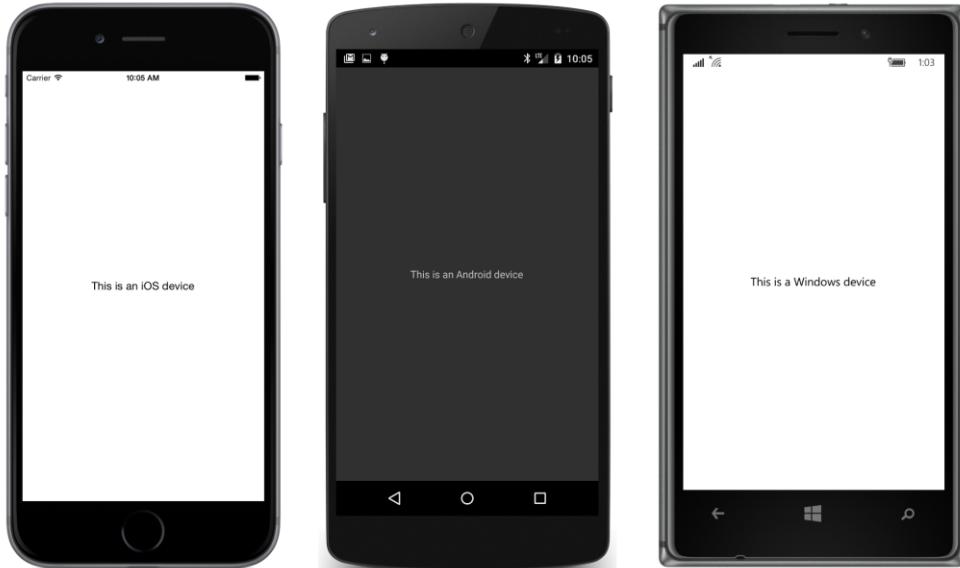
Se les puede dar diferentes nombres y manejar los tres nombres en el archivo de código subyacente mediante el uso de

Device.OnPlatform, pero la mejor solución es mantener el marcado específico de la plataforma lo más pequeño posible. En este ejemplo, todas las Etiqueta propiedades son las mismas excepto para Texto, lo que sólo el Texto la propiedad tiene que ser específico de la plataforma. Aquí está la versión de la PlatformSpecificLabels programa que se incluye con el código de ejemplo para este capítulo. Tiene una sola Etiqueta, y todo lo que es independiente de la plataforma a excepción de la Texto propiedad:

```
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns:x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " PlatformSpecificLabels.PlatformSpecificLabelsPage " >

    < Etiqueta x: Nombre = " DEVICELABEL "
        HorizontalOptions = " Centrar "
        VerticalOptions = " Centrar " >
        < label.text >
            < OnPlatform x: TypeArguments = " x: String " >
                iOS = " Este es un dispositivo iOS "
                Androide = " Se trata de un dispositivo Android "
                WinPhone = " Se trata de un dispositivo con Windows " />
            </ label.text >
        </ Etiqueta >
    </ Pagina de contenido >
```

Esto es lo que parece:



los Texto propiedad es la propiedad de contenido para Etiqueta, por lo que no es necesario el label.text etiquetas en el ejemplo anterior.

Esto funciona así:

```
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns:x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " PlatformSpecificLabels.PlatformSpecificLabelsPage " >
```

```
< Etiqueta x: Nombre = "DEVICELABEL" 
    HorizontalOptions = "Centrar" 
    VerticalOptions = "Centrar" >
< OnPlatform x: TypeArguments = "x: String" 
    iOS = "Este es un dispositivo iOS" 
    Androide = "Se trata de un dispositivo Android" 
    WinPhone = "Se trata de un dispositivo con Windows" />
</ Etiqueta >
</ Pagina de contenido >
```

puntos de vista basados en XAML personalizada

los **ScaryColorList** programa en el capítulo anterior aparece unos pocos colores en una **StackLayout** utilizando **Marco**, **BoxView**, y **Etiqueta**. Incluso con sólo tres colores, el marcado repetitivo estaba empezando a verse muy mal agüero. Desafortunadamente, no hay marcado XAML que duplique el C # para y mientras bucles, por lo que su opción es utilizar código para generar múltiples artículos similares, o para encontrar una mejor manera de hacerlo en el marcado.

En este libro, verá varias maneras para mostrar colores en XAML, y, finalmente, de una manera muy limpia y elegante para hacer este trabajo se hará evidente. Pero eso requiere algunos pasos más en Xamarin.Forms aprendizaje. Hasta entonces, vamos a estar buscando en algunos otros enfoques que le puede resultar útil en circunstancias similares.

Una estrategia consiste en crear una vista personalizada que tiene el único propósito de mostrar un color con un nombre y un cuadro de color. Y ya que estamos en ello, vamos a mostrar los valores RGB hexadecimales de los colores también. A continuación, puede utilizar esa vista personalizada en un archivo de página XAML para los colores individuales.

Lo que podría una referencia a una vista tal costumbre parecerse en XAML?

O la mejor pregunta es: ¿Cómo *me gusta* que se vea?

Si el marcado se veía algo así, la repetición no es malo en absoluto, y no mucho peor que la definición explícita de una serie de Color valores en código:

```
< StackLayout >
    < MyColorView Color = "rojo" />
    < MyColorView Color = "Verde" />
    < MyColorView Color = "Azul" />
    ...
</ StackLayout >
```

Bueno, en realidad, no va a ser exactamente así. **MyColorView** es obviamente una clase personalizada y no forma parte de la API Xamarin.Forms. Por lo tanto, no puede aparecer en el archivo XAML sin un prefijo de espacio de nombres que se define en una declaración de espacio de nombres XML.

Con este prefijo XML aplicado, no habrá ninguna confusión acerca de esta vista personalizada de ser parte de la API Xamarin.Forms, así que vamos a darle un nombre más digno de **ColorView** más bien que **MyColorView**.

esta hipotética ColorView clase es un ejemplo de una vista personalizada bastante fácil, ya que consiste solamente en puntos de vista existentes, específicamente Etiqueta, marco, y BoxView -arranged de una manera particular el uso de StackLayout. Xamarin.Forms define una visión diseñada específicamente para el propósito de la crianza de un arreglo de tales puntos de vista, y se llama ContentView. Me gusta ContentPage, ContentView tiene un Estafa-
tentView en el código, pero es más divertido hacerlo en XAML.

Vamos a poner juntos una solución llamada ColorViewList. Esta solución tendrá dos juegos de XAML y archivos de código subyacente, el primero de una clase llamada ColorViewListPage, que se deriva de Pagina de contenido (Como de costumbre), y el segundo para una clase llamada ColorView, que se deriva de ContentView.

Para crear el ColorView clase en Visual Studio, utilice el mismo procedimiento que cuando se añade una nueva página XAML a la ColorViewList proyecto: Haga clic en el nombre del proyecto en el Explorador de la solución, y seleccione Agregar> Nuevo elemento en el menú contextual. En el Agregar ítem nuevo diálogo, seleccione Visual C #> Multiplataforma a la izquierda y luego Página forma Xaml. Introduzca los ColorView.cs nombre. Pero de inmediato, antes de que se le olvida, entra en el archivo y cambiar el ColorView.xaml Pagina de contenido empezar y terminar a las etiquetas

ContentView. En el archivo ColorView.xaml.cs, cambiar la clase base a ContentView.

El proceso es un poco más fácil en Xamarin Studio. Desde el menú de herramientas para la ColorViewList proyecto, seleccione Agregar> Nuevo archivo. En el Archivo nuevo diálogo, seleccione formas a la izquierda y ContentView forma Xaml (no Formas ContentPage Xaml). Darle un nombre de ColorView.

También tendrá que crear un archivo XAML y el archivo de código subyacente para el ColorViewListPage clase, como de costumbre.

El archivo ColorView.xaml describe la disposición de los elementos de color individuales, pero sin ningún tipo de valores de los colores reales. En cambio, el BoxView y dos Etiqueta puntos de vista son nombres dados:

```
<ContentView xmlns = " http://xamarin.com/schemas/2014/forms "
             xmlns:x = " http://schemas.microsoft.com/winfx/2009/xaml "
             x:Class = " ColorViewList.ColorView " >

    <Marco OutlineColor = " Acento " >
        < StackLayout Orientación = " Horizontal " >
            < BoxView x: Nombre = " boxView "
                      WidthRequest = " 70 "
                      HeightRequest = " 70 " />

            < StackLayout >
                < Etiqueta x: Nombre = " colorNameLabel "
                          Tamaño de fuente = " Grande "
                          VerticalOptions = " CenterAndExpand " />

                < Etiqueta x: Nombre = " colorValueLabel "
                          VerticalOptions = " CenterAndExpand " />
            </ StackLayout >
        </ StackLayout >
    </ Marco >
</ ContentView >
```

En un programa de la vida real, usted tendrá un montón de tiempo más tarde para afinar las imágenes. Inicialmente, usted sólo quiere obtener todas las vistas guardadas en ese país.

Además de los efectos visuales, este ColorView clase tendrá una nueva propiedad para establecer el color. Esta propiedad se debe definir en el archivo de código subyacente. En un primer momento, parece razonable dar ColorView una propiedad denominada Color de tipo Color (como los anteriores XAML con snippet MyColorView parece sugerir). Pero el ColorView clase necesita para visualizar el color *nombre*, y no se puede obtener el nombre del color de una Color valor.

En cambio, tiene más sentido para definir una propiedad denominada Colormame de tipo cuerda. El archivo de código subyacente a continuación, puede utilizar la reflexión para obtener el campo estático de la Color clase correspondiente a ese nombre.

Pero, un momento: Xamarin.Forms incluye una pública ColorTypeConverter clase que el analizador XAML utiliza para convertir un nombre de color de texto como "Rojo" o "azul" en una Color valor. Por qué no tomar ventaja de eso?

Aquí está el archivo de código subyacente para ColorView. Define una Colormame propiedad con una conjunto de acceso que establece el Texto propiedad de la colorNameLabel al nombre del color, y utiliza a continuación, ColorType-Convertidor para convertir el nombre a una Color valor. Esta Color valor se utiliza entonces para establecer la Color propiedad de boxView y el Texto propiedad de la colorValueLabel a los valores RGB:

```
público clase parcial ColorView : ContentView
{
    cuerda colorName;
    ColorTypeConverter colorTypeConv = nuevo ColorTypeConverter ();

    público ColorView ()
    {
        InitializeComponent ();
    }

    public string Colormame
    {
        conjunto
        {
            // Establecer el nombre.
            colorName = valor;
            colorNameLabel.Text = valor;

            // Obtener el color real y establecer los otros puntos de vista.
            Color color = ( Color ) ColorTypeConv.ConvertFrom (colorName);
            boxView.Color = color;
            colorValueLabel.Text = Cuerda .Formato( "(0: X2) - (1: X2) - (2: X2)" ,
                ( Ent ) (255 * color.R),
                ( Ent ) (255 * color.G),
                ( Ent ) (255 * color.B));
        }
        obtener
        {
            regreso colorName;
        }
    }
}
```

```

    }
}

```

los ColorView la clase ha terminado. Ahora vamos a ver `ColorViewListPage`. El archivo debe enumerar múltiples `ColorViewListPage.xaml` `ColorView` casos, por lo que necesita una nueva declaración de espacio de nombres XML con un nuevo prefijo de espacio para hacer referencia al `ColorView` elemento.

los ColorView clase es parte de un mismo proyecto, `ColorViewListPage`. En general, los programadores utilizan un prefijo de espacio de nombres XML de local para estos casos. La nueva declaración de espacio aparece en el elemento raíz del archivo XAML (como los otros dos) con el siguiente formato:

```
xmins: locales = "clr-espacio de nombres: ColorViewList; montaje = ColorViewList"
```

En el caso general, una declaración de espacio de nombres XML personalizado para XAML debe especificar un tiempo de ejecución de lenguaje común (CLR) de espacio de nombres también se conoce como el espacio de nombres de .NET, y una asamblea. Las palabras clave para especificar estos son CLR-espacio de nombres y montaje. A menudo, el espacio de nombres CLR es el mismo que el montaje, como lo son en este caso, pero que no necesitan ser. Las dos partes están conectadas por un punto y coma.

Observe que dos puntos sigue CLR-espacio de nombres, pero un signo igual sigue montaje. Esta aparente contradicción es deliberada: el formato de la declaración de espacio de nombres se pretende imitar un URI encontrado en las declaraciones de espacio de nombres convencionales, en el que una de colon sigue al nombre de esquema URI.

Se utiliza la misma sintaxis para hacer referencia a objetos de bibliotecas de clases portátiles externos. La única diferencia en estos casos es que el proyecto necesita también una referencia a esa PCL externa. (Vas a ver un ejemplo en el capítulo 10, “Extensión de marcado XAML.”).

los local prefijo es común para el código en el mismo conjunto, y en ese caso la montaje parte no se requiere:

```
xmins: locales = "CLR-espacio de nombres: ColorViewList"
```

Para un archivo XAML en un PCL, puede incluir la montaje partes para hacer referencia algo en el mismo conjunto si se quiere, pero no es necesario. Para un archivo XAML en un SAP, sin embargo, debe *no* incluir la montaje parte para hacer referencia a una clase local porque no hay montaje asociado con un SAP. El código en el SAP es en realidad parte de los conjuntos de plataformas individuales, y los que todos tienen diferentes nombres.

Aquí está el XAML para el `ColorViewListPage` clase. El archivo de código subyacente contiene nada más allá de la `InitializeComponent` llamada:

```

< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    xmlns: locales = " CLR-espacio de nombres: ColorViewList "
    x: Class = " ColorViewList.ColorViewListPage " >

    < ContentPage.Padding >
        < OnPlatform x: TypeArguments = " Espesor "
            iOS = " 0, 20, 0, 0 " />
    </ ContentPage.Padding >

```

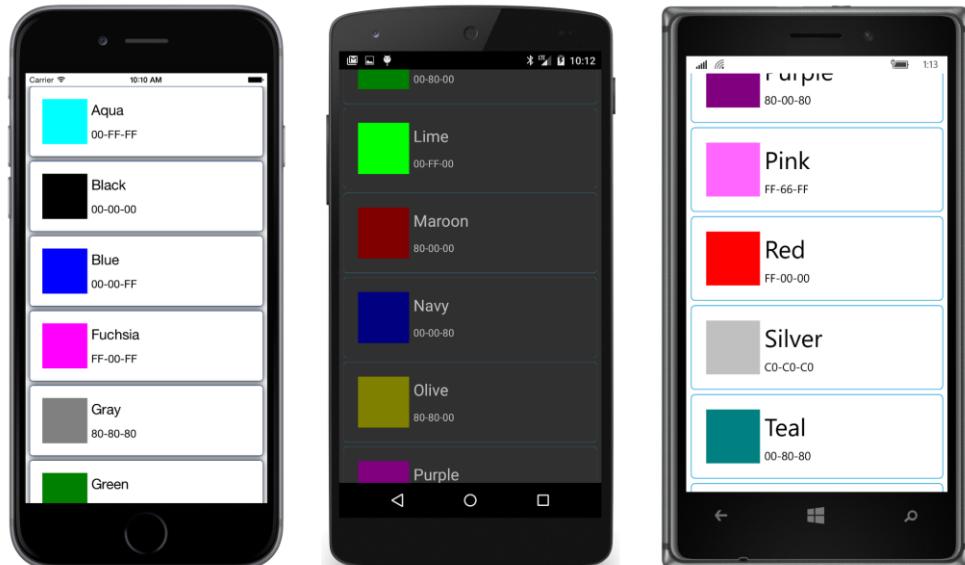
```

< ScrollView >
    < StackLayout Relleno = " 6, 0 " >
        < locales: ColorView Colormame = " Agua " />
        < locales: ColorView Colormame = " Negro " />
        < locales: ColorView Colormame = " Azul " />
        < locales: ColorView Colormame = " Fucsia " />
        < locales: ColorView Colormame = " gris " />
        < locales: ColorView Colormame = " Verde " />
        < locales: ColorView Colormame = " Lima " />
        < locales: ColorView Colormame = " Granate " />
        < locales: ColorView Colormame = " Armada " />
        < locales: ColorView Colormame = " Aceituna " />
        < locales: ColorView Colormame = " Púrpura " />
        < locales: ColorView Colormame = " Rosado " />
        < locales: ColorView Colormame = " rojo " />
        < locales: ColorView Colormame = " Plata " />
        < locales: ColorView Colormame = " Teal " />
        < locales: ColorView Colormame = " Blanco " />
        < locales: ColorView Colormame = " Amarillo " />
    </ StackLayout >
</ ScrollView >
</ Pagina de contenido >

```

Esto no es tan odiosa como el ejemplo anterior parece sugerir, y demuestra cómo se puede encapsular elementos visuales en sus propias clases basadas en XAML. Observe que el StackLayout es el hijo de una

ScrollView, por lo que la lista puede ser desplazado:



Sin embargo, hay un aspecto de la **ColorViewList** proyecto que no califica como una "mejor práctica". Es la definición de la Colormame bienes en ColorView. Esto debería ser implementado como una

BindableProperty objeto. Profundizando en objetos que pueden vincularse y propiedades enlazables es una alta prioridad y será explorado en el capítulo 11, "La infraestructura enlazable."

Eventos y controladores

Cuando puentes en un Xamarin.Forms Botón, se dispara una hecho clic evento. Puede crear una instancia de una Botón en XAML, pero el hecho clic controlador de eventos en sí debe residir en el archivo de código subyacente. Los Botón es sólo uno de un montón de puntos de vista que existen principalmente para generar eventos, por lo que el proceso de manejo de eventos es crucial para la coordinación de archivos XAML y código.

Instalación de un controlador de eventos para un evento en XAML es tan simple como la creación de una propiedad; es, de hecho, visualmente indistinguible de un valor de la propiedad. Los XamlKeypad proyecto es una versión de la XAML PersistentKeypad proyecto de Capítulo 6. Se ilustra la configuración de controladores de eventos en XAML y gastos de estos eventos en el archivo de código subyacente. También incluye la lógica para guardar las entradas del teclado cuando se termina el programa.

Si se echa un vistazo atrás en el código del constructor SimplestKeypadPage o PersistentKeyPadPage clases, verá un par de bucles para crear los botones que componen la parte numérica del teclado. Por supuesto, este es precisamente el tipo de cosas que no puede hacer en XAML, pero mira cuánto más limpio en el marcado XamlKeypadPage es cuando se compara con ese código:

```
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns:x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " XamlKeypad.XamlKeypadPage " >

    < StackLayout VerticalOptions = " Centrar "
        HorizontalOptions = " Centrar " >

        < Etiqueta x: Nombre = " displayLabel "
            Fuente = " Grande "
            VerticalOptions = " Centrar "
            HorizontalTextAlignment = " Fin " />

        < Botón x: Nombre = " backspaceButton "
            Texto = " & # X21E6; "
            Fuente = " Grande "
            Está habilitado = " Falso "
            hecho clic = " OnBackspaceButtonClicked " />

    < StackLayout Orientación = " Horizontal " >
        < Botón Texto = " 7 " styleId = " 7 " Fuente = " Grande "
            hecho clic = " OnDigitButtonClicked " />
        < Botón Texto = " 8 " styleId = " 8 " Fuente = " Grande "
            hecho clic = " OnDigitButtonClicked " />
        < Botón Texto = " 9 " styleId = " 9 " Fuente = " Grande "
            hecho clic = " OnDigitButtonClicked " />
    </ StackLayout >
```

```
< StackLayout Orientación = "Horizontal" >
    < Botón Texto = "4" styleId = "4" Fuente = "Grande" 
        hecho clic = "OnDigitButtonClicked" />
    < Botón Texto = "5" styleId = "5" Fuente = "Grande" 
        hecho clic = "OnDigitButtonClicked" />
    < Botón Texto = "6" styleId = "6" Fuente = "Grande" 
        hecho clic = "OnDigitButtonClicked" />
</ StackLayout >

< StackLayout Orientación = "Horizontal" >
    < Botón Texto = "1" styleId = "1" Fuente = "Grande" 
        hecho clic = "OnDigitButtonClicked" />
    < Botón Texto = "2" styleId = "2" Fuente = "Grande" 
        hecho clic = "OnDigitButtonClicked" />
    < Botón Texto = "3" styleId = "3" Fuente = "Grande" 
        hecho clic = "OnDigitButtonClicked" />
</ StackLayout >

< Botón Texto = "0" styleId = "0" Fuente = "Grande" 
    hecho clic = "OnDigitButtonClicked" />

</ StackLayout >
</ Pagina de contenido >
```

El archivo es mucho más corto de lo que hubiera sido si las tres propiedades de cada numérico Botón
sido formateada en tres líneas, pero éstos embalaje todos juntos hace que la uniformidad del margen de beneficio muy obvio y proporciona
claridad en lugar de la oscuridad.

La gran pregunta es la siguiente: ¿Qué prefieres mantener y modificar? El código en el Simplest-
KeypadPage o PersistentKeypadPage constructores o el marcado en el XamlKeypadPage XAML presentar?

Aquí está la captura de pantalla. Verá que estas teclas están dispuestas en orden calculadora en lugar de pedido telefónico:



El botón de retroceso tiene su hecho clic evento que se desarrolla a la OnBackspaceButtonClicked manipulador, mientras que los botones numéricos comparten el OnDigitButtonClicked entrenador de animales. Como se recordará, el styleId propiedad se utiliza a menudo para distinguir puntos de vista que comparten el mismo controlador de eventos, lo que significa que los dos controladores de eventos se pueden implementar en el archivo de código subyacente exactamente el mismo que en el programa de sólo código:

```

pública clase parcial XamlKeypadPage
{
    Aplicación aplicación = Solicitud .Corriente como Aplicación ;

    pública XamlKeypadPage ()
    {
        InitializeComponent ();

        displayLabel.Text = app.DisplayLabelText;
        backspaceButton.IsEnabled = displayLabel.Text!= nulo &&
            displayLabel.Text.Length> 0;
    }

    vacío OnDigitButtonClicked ( objeto remitente, EventArgs args)
    {
        Botón botón = ( Botón )remitente;
        displayLabel.Text += ( cuerda )Button.StyleId;
        backspaceButton.IsEnabled = cierto ;

        app.DisplayLabelText = displayLabel.Text;
    }

    vacío OnBackspaceButtonClicked ( objeto remitente, EventArgs args)
    {
        cuerda text = displayLabel.Text;
        displayLabel.Text = text.Substring (0, text.length - 1);
    }
}

```

```
backspaceButton.IsEnabled = displayLabel.Text.Length > 0;

app.DisplayLabelText = displayLabel.Text;
}

}
```

Parte del trabajo de la LoadFromXaml método llamado por InitializeComponent implica unir estos controladores de eventos a los objetos instanciados a partir del archivo XAML.

los XamarinKeypad proyecto también incluye el código que se agrega a la página y Aplicación en clases PersistentKeypad para guardar el texto teclado cuando se termina el programa. los Aplicación clase de XamarinKeypad es básicamente el mismo que el de PersistentKeypad.

Toque gestos

los Xamarin.Forms Botón responde a los toques de los dedos, pero en realidad se puede obtener el dedo grifos de cualquier clase que deriva de Ver, incluso Etiqueta, BoxView, y Marco. Estos eventos grifo no están incorporadas en el Ver clase, pero el Ver clase define una propiedad denominada GestureRecognizers. Grifos están habilitados mediante la adición de un objeto a esta GestureRecognizers colección. Una instancia de cualquier clase que deriva de GestureRecognizer Se pueden añadir a esta colección, pero sin duda el más útil es TapGestureRecognizer.

Aquí es cómo agregar una TapGestureRecognizer a una BoxView en código:

```
BoxView boxView = nuevo BoxView
{
    color = Color.Azul
};
TapGestureRecognizer tapGesture = nuevo TapGestureRecognizer();
tapGesture.Tapped += OnBoxViewTapped;
boxView.GestureRecognizers.Add (tapGesture);
```

TapGestureRecognizer también define una NumberOfTapsRequired propiedad con un valor predeterminado de 1. Juego de 2 a implementar dobles grifos.

Para generar aprovechado eventos, el Ver objeto debe tener su Esta habilitado propiedad establecida en cierto, sus EsVisible propiedad establecida en cierto (o no será visible en absoluto), y su InputTransparent propiedad establecida en falso. Estas son todas las condiciones predeterminadas.

los aprovechado manejador se parece a una hecho clic controlador para el Botón:

```
vacio OnBoxViewTapped ( objeto remitente, EventArgs args)
{
    ...
}
```

Como se sabe, el remitente argumento de un controlador de eventos es normalmente el objeto que desencadena el evento,

que en este caso sería el TapGestureRecognizer objeto. Eso no sería de mucha utilidad. En cambio, el remitente argumento de la aprovechado manejador es la vista siendo aprovechado, en este caso el BoxView.
Eso es mucho ¡más útil!

Me gusta Botón, TapGestureRecognizer también define Mando y CommandParameter propiedades; estos se utilizan cuando se implementa el patrón de diseño MVVM, y también son discutidos en un capítulo posterior.

TapGestureRecognizer también define propiedades con nombre TappedCallback y TappedCallback-Parámetro y un constructor que incluye una TappedCallback argumento. Todos ellos están obsoletos y no deben utilizarse.

En XAML, puede adjuntar una TapGestureRecognizer a una vista expresando el GestureRecognizer independiente de organizadores colección como un elemento de propiedad:

```
<BoxView Color = "Azul">
    <BoxView.GestureRecognizers>
        <TapGestureRecognizer aprovechado = "OnBoxViewTapped" />
    </BoxView.GestureRecognizers>
</BoxView>
```

Como de costumbre, el XAML es un poco más corto que el código equivalente.

Vamos a hacer un programa que está inspirado en uno de los primeros juegos de ordenador independientes.

La versión Xamarin.Forms de este juego se llama **MonkeyTap** porque es un juego de imitación. Contiene cuatro BoxView elementos, de color rojo, azul, amarillo y verde. Cuando el juego comienza, una de las BoxView Elementos parpadea y, a continuación, deben aprovechar ese BoxView. Ese BoxView destellos seguidos de nuevo por otro, y ahora tiene que tocar tanto en secuencia. Entonces esos dos destellos son seguidas por una tercera y así sucesivamente. (El original tenía sonar así, pero **MonkeyTap** no lo hace.) Es un juego bastante cruel porque no hay manera de ganar. El juego sólo sigue cada vez más difícil hasta que lo pierde.

El archivo MonkeyTapPage.xaml instancia los cuatro BoxView elementos y una Botón en el centro de la etiqueta "Comenzar".

```
<Página de contenido xmlns = "http://xamarin.com/schemas/2014/forms"
    xmlns:x = "http://schemas.microsoft.com/winfx/2009/xaml"
    x:Class = "MonkeyTap.MonkeyTapPage">

    <ContentPage.Padding>
        <OnPlatform x:TypeArguments = "Espesor"
            iOS = "0, 20, 0, 0" />
    </ContentPage.Padding>

    <StackLayout>
        <BoxView x:Nombre = "boxview0"
            VerticalOptions = "FillAndExpand">
            <BoxView.GestureRecognizers>
                <TapGestureRecognizer aprovechado = "OnBoxViewTapped" />
            </BoxView.GestureRecognizers>
        </BoxView>
    </StackLayout>

```

```

</ BoxView >

< BoxView x: Nombre = " boxview1 "
    VerticalOptions = " FillAndExpand " >
    < BoxView.GestureRecognizers >
        < TapGestureRecognizer aprovechado = " OnBoxViewTapped " />
    </ BoxView.GestureRecognizers >
</ BoxView >

< Botón x: Nombre = " startGameButton "
    Texto = " Empezar "
    Fuente = " Grande "
    HorizontalOptions = " Centrar "
    hecho clic = " OnStartGameButtonClicked " />

< BoxView x: Nombre = " boxview2 "
    VerticalOptions = " FillAndExpand " >
    < BoxView.GestureRecognizers >
        < TapGestureRecognizer aprovechado = " OnBoxViewTapped " />
    </ BoxView.GestureRecognizers >
</ BoxView >

< BoxView x: Nombre = " boxview3 "
    VerticalOptions = " FillAndExpand " >
    < BoxView.GestureRecognizers >
        < TapGestureRecognizer aprovechado = " OnBoxViewTapped " />
    </ BoxView.GestureRecognizers >
</ BoxView >

</ StackLayout >
</ Página de contenido >

```

Los cuatro BoxView elementos que aquí tienen una TapGestureRecognizer unidos, pero aún no se asignan colores. Eso lo maneja en el archivo de código subyacente debido a que los colores no se mantendrá constante. Los colores tienen que ser cambiado para el efecto de parpadeo.

El archivo de código subyacente comienza con algunas constantes y campos variables. (Se dará cuenta de que una de ellas está marcada como protegida, que en el siguiente capítulo, una clase se deriva de éste y requieren el acceso a este campo. Algunos métodos se definen como protegidos también.).

```

pública clase parcial MonkeyTapPage
{
    const int sequenceTime = 750;                                // en ms
    protegida int const flashDuration = 250;

    const doble offLuminosity = 0,4;                            // un poco más tenue
    const doble onLuminosity = 0,75;                            // mucho más brillante

    BoxView [] BoxViews;
    Color [] = {Colores Color .Rojo, Color .Azul, Color .Amarillo, Color .Verde };

    Lista < En t > Secuencia = nuevo Lista < En t > ();
    En t sequenceIndex;
    bool awaitingTaps;
    bool gameEnded;
}

```

```

Aleatorio = aleatorios nuevo Aleatorio ();

público MonkeyTapPage ()
{
    InitializeComponent ();
    boxViews = nuevo BoxView [] {Boxview0, boxview1, boxview2, boxview3};
    InitializeBoxViewColors ();
}

vacío InitializeBoxViewColors ()
{
    para (En i; index = 0; índice <4; índice++)
        boxViews [índice].Color = colores [índice].WithLuminosity (offLuminosity);
}
...
}

```

El constructor pone los cuatro BoxView elementos de una matriz; esto les permite ser referenciados por un índice simple que tiene valores de 0, 1, 2, y 3. El InitializeBoxViewColors método establece toda la Caja-Ver elementos a su estado nonflashed ligeramente atenuado.

El programa está a la espera de que el usuario pulse el **Empezar** botón para iniciar el primer juego. Lo mismo Botón se ocupa de las repeticiones, lo que incluye una inicialización redundante de la BoxView colores. Los Botón manejador prepara también para la construcción de la secuencia de Flashed BoxView elementos por despejar el SE-cuencia lista y vocación StartSequence:

```

público clase parcial MonkeyTapPage
{
    ...
    protected void OnStartGameButtonClicked (objeto remitente, EventArgs args)
    {
        gameEnded = falso ;
        startGameButton.isVisible = falso ;
        InitializeBoxViewColors ();
        sequence.Clear ();
        StartSequence ();
    }

    vacío StartSequence ()
    {
        sequence.Add (random.Next (4));
        sequenceIndex = 0;
        Dispositivo .StartTimer (Espacio de tiempo .FromMilliseconds (sequenceTime), OnTimerTick);
    }
    ...
}

```

StartSequence añade un nuevo número entero aleatorio a la secuencia Lista, inicializa sequenceIndex a 0 y se inicia el temporizador.

En el caso normal, el controlador de temporizador tic es llamado para cada índice en el secuencia enumerar y ocasiona que el BoxView a parpadear con una llamada a FlashBoxView. Los rendimientos de controlador temporizador falso

cuando la secuencia está en un extremo, indicando también configurando awaitingTaps que es el momento para el usuario para imitar la secuencia:

```
pública clase parcial MonkeyTapPage
{
    ...
    bool OnTimerTick ()
    {
        Si (GameEnded)
            falso retorno ;

        FlashBoxView (secuencia [sequencelIndex]);
        sequencelIndex++;
        awaitingTaps = sequencelIndex == sequence.Count;
        sequencelIndex = awaitingTaps? 0: sequencelIndex;
        regreso ! awaitingTaps;
    }

    virtual void protegida FlashBoxView ( En t índice)
    {
        boxViews [índice] .Color = colores [índice] .WithLuminosity (onLuminosity);
        Dispositivo .StartTimer ( Espacio de tiempo .FromMilliseconds (flashDuration), () =>
        {
            Si (GameEnded)
                falso retorno ;

            boxViews [índice] .Color = colores [índice] .WithLuminosity (offLuminosity);
            falso retorno ;
        });
    }
    ...
}
}
```

El flash es sólo un cuarto de segundo de duración. los FlashBoxView método establece por primera vez la luminosidad de un color brillante y crea un temporizador de "one-shot", llamado así por el método de devolución de llamada de temporizador (aquí expresado como una función lambda) devuelve falso y se apaga el temporizador después de restaurar la luminosidad del color.

los aprovechado controlador para el BoxView elementos ignora el grifo si el juego ya ha terminado (que sólo ocurre con un error por parte del usuario), y el juego termina si el usuario toca prematuramente sin esperar a que el programa para ir a través de la secuencia. De lo contrario, sólo se compara el golpecitos BoxView con la siguiente en la secuencia, que parpadea BoxView Si es correcto, o el juego termina si no:

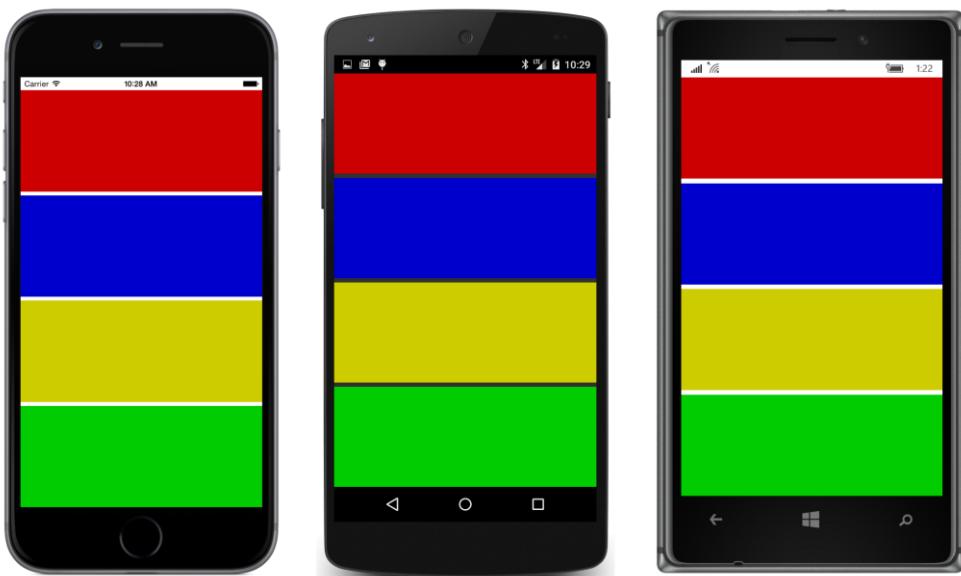
```
pública clase parcial MonkeyTapPage
{
    ...
    protected void OnBoxViewTapped ( objeto remitente, EventArgs args)
    {
        Si (GameEnded)
            regreso ;

        Si (! AwaitingTaps)
```

```
{  
    EndGame ();  
    regreso ;  
}  
  
BoxView tappedBoxView = ( BoxView )remitente;  
En t index = Formación .IndexOf (boxViews, tappedBoxView);  
  
Si (Índice! = Secuencia [sequencelIndex])  
{  
    EndGame ();  
    regreso ;  
}  
  
FlashBoxView (index);  
  
sequencelIndex ++;  
awaitingTaps = sequencelIndex <sequence.Count;  
  
Si (! AwaitingTaps)  
    StartSequence ();  
}  
  
protegido virtual void EndGame ()  
{  
    gameEnded = cierto ;  
  
    para ( En t index = 0; índice <4; índice ++)  
        boxViews [índice] .Color = Color .Gris;  
  
    startGameButton.Text = "¿Inténtalo de nuevo?";  
    startGameButton.isVisible = cierto ;  
}  
}
```

Si el usuario se las arregla para “mono” la secuencia de todo el camino a través, otra llamada a StartSequence añade un nuevo índice para la secuencia lista y empezará a reproducir esa nueva. Eventualmente, sin embargo, habrá una llamada a EndGame, los colores que todas las cajas grises para enfatizar el final, y vuelve a habilitar la Botón para tener la oportunidad de intentarlo de nuevo.

Aquí está el programa después de la Botón se ha hecho clic y oculto:



Sé que sé. El juego es un verdadero lastre sin sonido.

Vamos a tomar la oportunidad en el siguiente capítulo de arreglar eso.

Capítulo 9

Llamadas a la API específicas de la plataforma

Ha surgido una emergencia. Cualquiera que juegue con el **MonkeyTap** juego desde el capítulo anterior rápidamente se llega a la conclusión de que se necesita desesperadamente una mejora muy básica, y simplemente no se puede permitir que existir sin él.

MonkeyTap necesidades de sonido.

No necesita muy sofisticados sonoras sólo pequeños pitidos para acompañar a los destellos de los cuatro **BoxView** elementos. Pero la API **Xamarin.Forms** no admite sonido, por lo que el sonido no es algo que podemos añadir a **MonkeyTap** con sólo un par de llamadas a la API. Apoyando sonido requiere ir un poco más allá **Xamarin.Forms** para hacer uso de las instalaciones de generación de sonido específicas de la plataforma. Encontrar la manera de hacer que los sonidos en iOS, Android y Windows Phone es bastante difícil. Pero, ¿cómo un programa de **Xamarin.Forms** a continuación, hacer llamadas en las plataformas individuales?

Antes de abordar las complejidades de sonido, vamos a examinar los diferentes enfoques para realizar llamadas de API específicas de la plataforma con un ejemplo mucho más simple. Los tres primeros programas cortos que se muestran en este capítulo son funcionalmente idénticos: pantalla Todos ellos dos pequeños elementos de información suministrados por el sistema operativo de la plataforma subyacente que revelan el modelo del dispositivo que ejecuta el programa y la versión del sistema operativo.

Preprocesamiento en el Proyecto activo compartido

Como se vio en el capítulo 2, "Anatomía de una aplicación," se puede utilizar ya sea un proyecto compartido de Activos (SAP) o una biblioteca de clases portátil (PCL) para el código que es común a las tres plataformas. Un SAP contiene archivos de código que son compartidos entre los proyectos de plataforma, mientras que un PCL encierra el código común en una biblioteca que es accesible sólo a través de los tipos públicos.

El acceso a APIs de la plataforma de un proyecto activo compartido es un poco más sencilla que la de una biblioteca de clases portátil porque se trata de herramientas de programación más tradicionales, por lo que vamos a tratar de que el enfoque en primer lugar. Puede crear una solución **Xamarin.Forms** con un SAP mediante el proceso descrito en el capítulo 2. A continuación, puede añadir un XAML-basado Pagina de contenido la clase a la SAP de la misma forma que se agregan uno a un PCL.

Aquí está el archivo XAML para un proyecto que muestra información de la plataforma, llamada **PlatInfoSap1**:

```
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " PlatInfoSap1.PlatInfoSap1Page " >

    < StackLayout Relleno = " 20 " >
```

```

< StackLayout VerticalOptions = " CenterAndExpand " >
    < Etiqueta Texto = " Modelo de dispositivo: " />

    < ContentView Relleno = " 50, 0, 0, 0 " >
        < Etiqueta x: Nombre = " modelLabel "
            Tamaño de fuente = " Grande "
            FontAttributes = " Negrita " />
    </ ContentView >
</ StackLayout >

< StackLayout VerticalOptions = " CenterAndExpand " >
    < Etiqueta Texto = " Versión del sistema operativo: " />

    < ContentView Relleno = " 50, 0, 0, 0 " >
        < Etiqueta x: Nombre = " versionLabel "
            Tamaño de fuente = " Grande "
            FontAttributes = " Negrita " />
    </ ContentView >
</ StackLayout >
</ StackLayout >
</ Pagina de contenido >

```

El archivo de código subyacente debe establecer la Texto propiedades para modelLabel y versionLabel.

archivos de código en un proyecto activo compartido son extensiones del código en las plataformas individuales. Esto significa que el **código en el SAP puede hacer uso del preprocesador de C # directivas # Si, #elif, #más, y # terminara si con símbolos condicional-compilación definidos para las tres plataformas, como se demuestra en los capítulos 2 y 4. Estos símbolos son:**

- __IOS__ para iOS
- __ANDROIDE__ para Android
- WINDOWS_UWP para la plataforma de Windows universal
- WINDOWS_APP para Windows 8.1
- WINDOWS_PHONE_APP para Windows Phone 8.1

Las APIs necesarios para obtener la información del modelo y la versión son, por supuesto, diferente para las tres plataformas:

- Para iOS, utilice el UIDevice clase en el UIKit espacio de nombres.
- Para Android, utilizar varias propiedades de la Construir clase en el Android.OS espacio de nombres.
- Para las plataformas Windows, utilice el EasClientDeviceInformation clase en el Gamar-dows.Security.ExchangeActiveSyncProvisioning espacio de nombres.

Aquí está el archivo de código subyacente PlatInfoSap1.xaml.cs que muestra cómo modelLabel y versionLabel se establecen basándose en los símbolos de compilación condicional:

utilizando Sistema;

utilizando Xamarin.Forms;

Si __IOS__

utilizando UIKit;

elif __ANDROIDE__

utilizando Android.OS;

elif WINDOWS_APP || WINDOWS_PHONE_APP || WINDOWS_UWP
utilizando Windows.Security.ExchangeActiveSyncProvisioning;

terminara si

espacio de nombres PlatInfoSap1

{

público clase parcial PlatInfoSap1Page : Pagina de contenido

{

público PlatInfoSap1Page ()

{

InitializeComponent ();

Si __IOS__

UIDevice device = nuevo UIDevice();

modelLabel.Text = device.Model.ToString();

versionLabel.Text = Cuerda .Formato("{0} {1}" , Device.SystemName,

device.SystemVersion);

elif __ANDROIDE__

modelLabel.Text = Cuerda .Formato("{0} {1}" , Build.Manufacturer,

Build.Model);

versionLabel.Text = Build.VERSION.Release.ToString();

elif WINDOWS_APP || WINDOWS_PHONE_APP || WINDOWS_UWP

EasClientDeviceInformation devinfo = nuevo EasClientDeviceInformation();

modelLabel.Text = Cuerda .Formato("{0} {1}" , DevInfo.SystemManufacturer,

devInfo.SystemProductName);

versionLabel.Text = devInfo.OperatingSystem;

terminara si

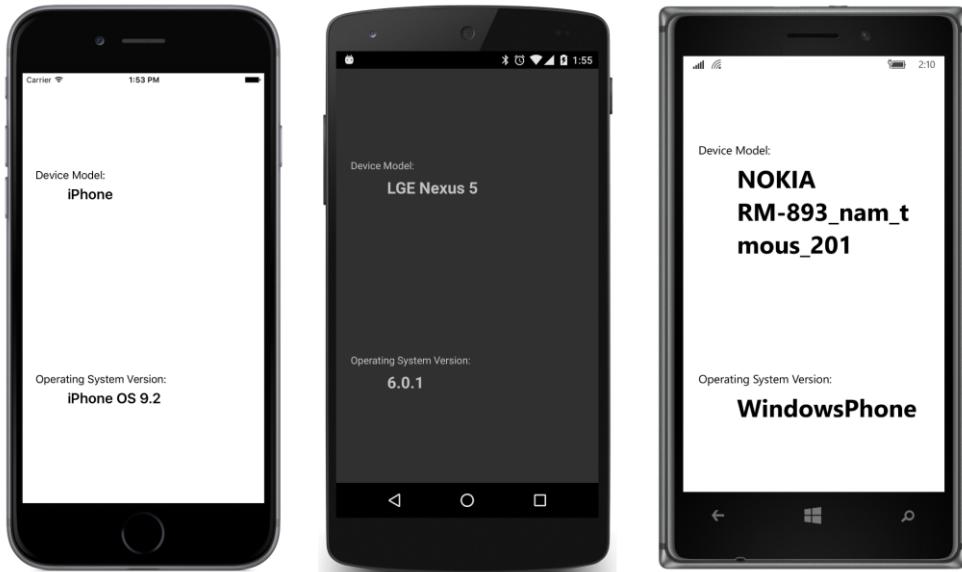
}

}

}

Tenga en cuenta que estas directivas del preprocesador se utilizan para seleccionar diferentes utilizando directivas, así como para realizar llamadas a las API específicas de la plataforma. En un programa tan simple como esto, simplemente podría incluir los espacios de nombres con los nombres de las clases, pero para los bloques más largos de código, es probable que desee aquellos utilizando directivas.

Y, por supuesto funciona:



La ventaja de este enfoque es que usted tiene todo el código para las tres plataformas en un solo lugar. Sin embargo, las directivas del preprocesador en el listado de código se-seamos sinceros, bastante feo, y Harken de nuevo a una época muy anterior en la programación. Uso de las directivas del preprocesador puede no parecer tan malo para llamadas cortas y menos frecuentes como este ejemplo, pero en un programa más amplio que va a necesitar para hacer malabares bloques de código específico de la plataforma y de código compartido, y la multitud de las directivas de preprocesador puede convertirse fácilmente en confuso . directivas del preprocesador se deben utilizar para pequeñas correcciones y elementos generalmente no es tan estructurales en la aplicación.

Vamos a intentar otro enfoque.

clases paralelas y el Proyecto activo compartido

Aunque el proyecto activo compartido es una extensión de los proyectos de plataforma, la relación va en ambos sentidos: al igual que un proyecto de plataforma puede hacer llamadas en código en un proyecto activo compartido, el SAP puede hacer llamadas en los proyectos de plataforma individuales.

Esto significa que podemos restringir las llamadas a la API específicas de la plataforma a clases en los proyectos de plataforma individuales. Si los nombres y espacios de nombres de estas clases en los proyectos de plataforma son los mismos, entonces el código en el SAP puede acceder a estas clases de manera transparente, independiente de la plataforma.

En el **PlatInfoSap2** solución, cada uno de los cinco proyectos de plataforma tiene una clase llamada **PlatformInfo** que contiene dos métodos que devuelven cierta objetos, nombrados **getModel** y **GetVersion**. Aquí está la versión de esta clase en el proyecto IOS:

utilizando Sistema;

```
utilizando UIKit;

espacio de nombres PlatInfoSap2
{
    clase pública PlatformInfo
    {
        UIDevice device = nuevo UIDevice ();

        public string GetModel ()
        {
            regreso device.Model.ToString ();
        }

        public string GetVersion ()
        {
            regreso Cuerda .Formato( "{0} {1}" , Device.SystemName,
                                     device.SystemVersion);
        }
    }
}
```

Observe el nombre de espacio de nombres. A pesar de las otras clases en este proyecto utilizan el IOS PlatInfo-Sap2.iOS espacio de nombres, el espacio de nombres para esta clase es sólo PlatInfoSap2. Esto permite que el SAP para acceder a esta clase directamente sin detalles específicos de la plataforma.

Aquí está la clase de al lado en el proyecto Android. Mismo espacio de nombres, el mismo nombre de la clase, y los mismos nombres de métodos, pero diferentes implementaciones de estos métodos que utilizan llamadas a la API de Android:

```
utilizando Sistema;
utilizando Android.OS;

espacio de nombres PlatInfoSap2
{
    clase pública PlatformInfo
    {
        public string GetModel ()
        {
            regreso Cuerda .Formato( "{0} {1}" , Construir .Fabricante,
                                     Construir .Modelo);
        }

        public string GetVersion ()
        {
            regreso Construir .VERSIÓN .Release.ToString ();
        }
    }
}
```

Y aquí está la clase que existe en tres ejemplares idénticos en los tres proyectos de Windows y Windows Phone:

```
utilizando Sistema;
utilizando Windows.Security.ExchangeActiveSyncProvisioning;
```

```

espacio de nombres PlatInfoSap2
{
    clase pública PlatformInfo
    {
        EasClientDeviceInformation devinfo = nuevo EasClientDeviceInformation();

        public string GetModel ()
        {
            regreso Cuerda .Formato( "{0} {1}" , DevInfo.SystemManufacturer,
                                     devInfo.SystemProductName);
        }

        public string GetVersion ()
        {
            regreso devInfo.OperatingSystem;
        }
    }
}

```

El archivo XAML en el **PlatInfoSap2** proyecto es básicamente el mismo que el de **PlatInfoSap1** proyecto. El archivo de código subyacente es considerablemente más simple:

```

utilizando Sistema;
utilizando Xamarin.Forms;

espacio de nombres PlatInfoSap2
{
    público clase parcial PlatInfoSap2Page : Pagina de contenido
    {
        público PlatInfoSap2Page ()
        {
            InitializeComponent ();

            PlatformInfo platformInfo = nuevo PlatformInfo ();
            modelLabel.Text = platformInfo.GetModel ();
            versionLabel.Text = platformInfo.GetVersion ();
        }
    }
}

```

La versión particular de **PlatformInfo** al que hace referencia la clase es el que está en el proyecto compilado. Es casi como si nos hemos definido un poco de extensión a **Xamarin.Forms** que reside en los proyectos de plataforma individuales.

DependencyService y la biblioteca de clases portátil

¿Puede la técnica ilustrada en la **PlatInfoSap2** programa se implementará en una solución con una biblioteca de clases portátil? Al principio, no parece posible. Aunque los proyectos de aplicación hacer llamadas a las bibliotecas de todo el tiempo, las bibliotecas en general, no pueden realizar llamadas a las aplicaciones, excepto en el contexto de los acontecimientos o

funciones de devolución de llamada. El ligamiento cruzado posterior empaquetado con una versión independiente del dispositivo de .NET y cerrada a cal y canto con capacidad única de ejecutar código en sí mismo u otros PCLs que podría hacer referencia.

Pero, un momento: Cuando una aplicación se está ejecutando Xamarin.Forms, se puede utilizar .NET reflexión para tener acceso a su propia asamblea y cualesquiera otros conjuntos en el programa. Esto significa que el código en el PCL puede utilizar la reflexión para acceder a las clases que existen en el conjunto de plataforma desde la cual se hace referencia a la PCL. Estas clases deben ser definidas como público, por supuesto, pero eso es casi el único requisito.

Antes de empezar a escribir código que explota esta técnica, usted debe saber que esta solución ya existe en la forma de una clase llamada **Xamarin.Forms DependencyService**. Esta clase utiliza .NET reflexión para buscar a través de todos los otros conjuntos en la aplicación, incluyendo el conjunto de plataforma en particular en sí, y proporcionar acceso al código específico de la plataforma.

El uso de **DependencyService** se ilustra en la **DisplayPlatformInfo** solución, que utiliza una biblioteca de clases portátil para el código compartido. Se empieza el proceso de utilización **DependencyService** mediante la definición de un tipo de interfaz en el proyecto PCL que declara las firmas de los métodos que desea implementar en los proyectos de plataforma. A continuación se **IPlatformInfo**:

```
espacio de nombres DisplayPlatformInfo
{
    público interfaz IPlatformInfo
    {
        cuerda GetModel ();
        cuerda GetVersion ();
    }
}
```

Usted ha visto esos dos métodos antes. Son los mismos dos métodos implementados en el **Plano-FormInfo** clases en los proyectos de plataforma en **PlatInfoSap2**.

De una manera muy similar a **PlatInfoSap2**, los tres proyectos de plataforma en **DisplayPlatformInfo** Ahora debe tener una clase que implementa la **IPlatformInfo** interfaz. Aquí está la clase en el proyecto de iOS, llamada **PlatformInfo**:

```
utilizando Sistema;
utilizando UIKit;
utilizando Xamarin.Forms;

[ montaje : Dependencia ( tipo de (DisplayPlatformInfo.iOS. PlatformInfo ))]

espacio de nombres DisplayPlatformInfo.iOS
{
    clase pública PlatformInfo : IPlatformInfo
    {
        UIDevice device = nuevo UIDevice ();

        public string GetModel ()
        {
            regreso device.Model.ToString ();
        }
    }
}
```

```

        }

        public string GetVersion ()
        {
            regreso Cuerda .Formato( "{0} {1}" , Device.SystemName,
                device.SystemVersion);
        }
    }
}

```

Esta clase no se hace referencia directamente desde el PCL, por lo que el nombre de espacio de nombres puede ser cualquier cosa que desee. Aquí se configura en el mismo espacio de nombres como el otro código en el proyecto IOS. El nombre de la clase también puede ser cualquier cosa que desee. Cualquiera que sea lo que sea, sin embargo, la clase debe implementar explícitamente el IPlatformInfo interfaz definida en el PCL:

Clase pública PlatformInfo : IPlatformInfo

Por otra parte, esta clase se debe hacer referencia en un atributo especial fuera del bloque de espacio de nombres. Lo verá en la parte superior del archivo después de la utilizando directivas:

```
[ montaje : Dependencia ( tipo de (DisplayPlatformInfo.iOS. PlatformInfo ))]
```

los DependencyAttribute clase que define este Dependencia atributo es parte de Xamarin.Forms y se utiliza específicamente en relación con DependencyService. El argumento es una Tipo objeto de una clase en el proyecto de plataforma que está disponible para el acceso de la PCL. En este caso, es este PlatformInfo clase. Este atributo está unido al conjunto de plataforma en sí, por lo que la ejecución de código en el PCL no tiene que buscar por toda la biblioteca para encontrarlo.

Aquí está la versión para Android de PlatformInfo:

```

utilizando Sistema;
utilizando Android.OS;
utilizando Xamarin.Forms;

[ montaje : Dependencia ( tipo de (DisplayPlatformInfo.Droid. PlatformInfo ))]

espacio de nombres DisplayPlatformInfo.Droid
{
    clase pública PlatformInfo : IPlatformInfo
    {
        public string GetModel ()
        {
            regreso Cuerda .Formato( "{0} {1}" , Construir .Fabricante,
                Construir .Modelo);
        }

        public string GetVersion ()
        {
            regreso Construir .VERSIÓN .Release.ToString ();
        }
    }
}

```

Y aquí está el uno para el proyecto uwp:

```
utilizando Sistema;
utilizando Windows.Security.ExchangeActiveSyncProvisioning;
utilizando Xamarin.Forms;

[ montaje : Dependencia ( tipo de (DisplayPlatformInfo.UWP, PlatformInfo ))]

espacio de nombres DisplayPlatformInfo.UWP
{
    clase pública PlatformInfo : IPlatformInfo
    {
        EasClientDeviceInformation devInfo = nuevo EasClientDeviceInformation();

        public string GetModel ()
        {
            regreso Cuerda .Formato( "{0} {1}" , DevInfo.SystemManufacturer,
                                    devInfo.SystemProductName);
        }

        public string GetVersion ()
        {
            regreso devInfo.OperatingSystem;
        }
    }
}
```

Las ventanas 8.1 y 8.1 de Windows Phone proyectos tienen archivos similares que se diferencian sólo por el espacio de nombres.

Código en el PCL puede entonces tener acceso a la implementación de la plataforma en particular de IPlatform- información mediante el uso de la DependencyService clase. Esta es una clase estática con tres métodos públicos, el más importante de ellos se llama Obtener. Obtener es un método genérico cuyo argumento es la interfaz que ha definido, en este caso IPlatformInfo.

```
IPlatformInfo platformInfo = DependencyService .get < IPlatformInfo > ();
```

los Obtener método devuelve una instancia de la clase específica de la plataforma que implemente la IPlatform- información interfaz. A continuación, puede utilizar este objeto para realizar llamadas específicas de la plataforma. Esto se demuestra en el archivo de código subyacente para el **DisplayPlatformInfo** proyecto:

```
espacio de nombres DisplayPlatformInfo
{
    público clase parcial DisplayPlatformInfoPage : Pagina de contenido
    {
        público DisplayPlatformInfoPage ()
        {
            InitializeComponent ();

            IPlatformInfo platformInfo = DependencyService .get < IPlatformInfo > ();
            modelLabel.Text = platformInfo.GetModel ();
            versionLabel.Text = platformInfo.GetVersion ();
        }
    }
}
```

```
}
```

DependencyService almacena en caché las instancias de los objetos que obtiene a través de la Obtener método. Esto acelera usos posteriores de Obtener y también permite que las implementaciones de la plataforma de la interfaz para mantener el estado: cualquier campo y propiedades en las implementaciones de la plataforma serán preservados a través de múltiples Obtener llamadas. Estas clases también pueden incluir eventos o implementar métodos de devolución de llamada.

DependencyService requiere sólo un poco más de riesgo que el enfoque se muestra en la PlatInfoSap2 proyecto y es un poco más estructurada porque las clases de plataforma individuales implementan una interfaz definida en código compartido.

DependencyService no es la única manera de poner en práctica las llamadas específicas de la plataforma en un PCL. desarrolladores de aventura puede ser que desee utilizar las técnicas de la dependencia de la inyección para configurar el PCL para hacer llamadas en los proyectos de plataforma. Pero DependencyService Es muy fácil de usar, y elimina la mayoría de razones para utilizar un proyecto activo compartido en una aplicación Xamarin.Forms.

la generación de sonido específico de la plataforma

Ahora, para el verdadero objetivo de este capítulo para dar sonido a **MonkeyTap**. Las tres plataformas soportan las API que permiten a un programa para generar y reproducir formas de onda de audio de forma dinámica. Este es el enfoque adoptado por el **MonkeyTapWithSound** programa.

archivos de música comerciales a menudo se comprimen en formatos tales como MP3. Pero cuando un programa está generando algorítmicamente formas de onda, un formato no comprimido es mucho más conveniente. La técnica de la que más básica es apoyado por las tres plataformas-se llama modulación por impulsos codificados o PCM. A pesar del nombre de fantasía, es bastante simple, y es la técnica utilizada para almacenar sonido en discos compactos de música.

Una forma de onda PCM es descrito por una serie de muestras a una velocidad constante, conocido como la velocidad de muestreo. CDs de música utilizan un índice de nivel de 44.100 muestras por segundo. Los archivos de audio generados por los programas de ordenador a menudo utilizan una frecuencia de muestreo de la mitad (22.050) o una cuarta parte (11.025) si no se requiere alta calidad de audio. La frecuencia más alta que puede ser grabada y reproducida es la mitad de la frecuencia de muestreo.

Cada muestra es un tamaño fijo que define la amplitud de la forma de onda en ese punto en el tiempo. Las muestras de un CD de música se firman valores de 16 bits. Las muestras de 8 bits son comunes cuando la calidad del sonido no importa tanto. Algunos entornos compatibles con los valores de coma flotante. muestras múltiples se pueden acomodar estéreo o cualquier número de canales. Para efectos de sonido simples en los dispositivos móviles, sonido monoaural es a menudo bien.

El algoritmo de generación de sonido en **MonkeyTapWithSound** está codificado para muestras monoaurales de 16 bits, pero la tasa de muestreo se especifica por una constante y se puede cambiar fácilmente.

Ahora que ya sabe cómo DependencyService funciona, vamos a examinar el código añadido a Mono-

Grifo para convertirlo en **MonkeyTapWithSound**, y vamos a ver desde la parte superior hacia abajo. Para evitar reproducir una gran cantidad de código, el nuevo proyecto contiene enlaces a los archivos y MonkeyTap.xaml MonkeyTap.xaml.cs en el **MonkeyTap** proyecto.

En Visual Studio, puede añadir elementos a proyectos como enlaces a los archivos existentes seleccionando **Añadir > elemento existente** En el menú proyecto. A continuación, utilice el **Agregar elemento existente** de diálogo para navegar hasta el archivo. Escoger

Agregar Enlace desde el menú desplegable en el **Añadir** botón.

En Xamarin Studio, seleccione **Añadir > Añadir archivos** desde el menú de herramientas del proyecto. Después de abrir el archivo o archivos, una **Agregar archivo a la carpeta** cuadro de alerta aparece. Escoger **Añadir un enlace al archivo**.

Sin embargo, después de tomar estos pasos en Visual Studio, también era necesario editar manualmente el archivo **MonkeyTapWithSound.csproj** para cambiar el archivo a un **MonkeyTapPage.xaml EmbeddedResource** y el **Generador a MSBuild: UpdateDesignTimeXaml**. También un **Depende de** etiqueta se añade al archivo **MonkeyTapPage.xaml.cs** hacer referencia al archivo **MonkeyTapPage.xaml**. Esto hace que el archivo de código subyacente a tener una sangría bajo el archivo XAML en la lista de archivos.

los MonkeyTapWithSoundPage clase, entonces se deriva de la MonkeyTapPage clase. Aunque el MonkeyTapPage clase se define por un archivo XAML y un archivo de código subyacente, MonkeyTapWithSoundPage es único código. Cuando una clase se deriva de esta forma, los controladores de eventos en el archivo original de código subyacente para los eventos en el archivo XAML deben ser definidas como protegido, y este es el caso.

los MonkeyTap clase también define una flashDuration constante protegido, y dos métodos se definieron como protegido y virtual. los MonkeyTapWithSoundPage anula estos dos métodos para llamar a un método estático denominado SoundPlayer.PlaySound:

```
espacio de nombres MonkeyTapWithSound
{
    clase MonkeyTapWithSoundPage : MonkeyTap. MonkeyTapPage
    {
        const int errorDuration = 500;

        // séptima disminuida en 1ra inversión: C, Eb, F #, A
        doble [] Frecuencias = {523.25, 622.25, 739.99, 880};

        protegido override void BlinkBoxView ( En t índice)
        {
            SoundPlayer .PlaySound (frecuencias [índice], flashDuration);
            base .BlinkBoxView (index);
        }

        protegido override void EndGame ()
        {
            SoundPlayer .PlaySound (65.4, errorDuration);
            base .EndGame ();
        }
    }
}
```

los SoundPlayer.PlaySound método acepta una frecuencia y una duración en milisegundos. Todo lo demás, el volumen, la composición armónica del sonido, y cómo se genera el sonido, es la responsabilidad del Reproducir sonido método. Sin embargo, este código hace una suposición implícita de que

SoundPlayer.PlaySound devuelve inmediatamente y no esperar a que el sonido de juego para completar. Afortunadamente, las tres plataformas compatibles con las API de generación de sonido que se comportan de esta manera.

los SoundPlayer clase con el Reproducir sonido método estático es parte de la MonkeyTapWithSound

proyecto PCL. La responsabilidad de este método es definir una matriz de los datos PCM para el sonido. El tamaño de esta matriz se basa en la tasa de muestreo y la duración. los para bucle calcula muestras que definen una onda triangular de la frecuencia solicitada:

```
espacio de nombres MonkeyTapWithSound
{
    clase SoundPlayer
    {
        const int samplingRate = 22050;

        // Hard-codificados para monoaural, PCM de 16 bits por muestra
        public static void Reproducir sonido( doble frecuencia = 440, Ent t duración = 250)
        {
            corto [] shortBuffer = nuevo corto [SamplingRate * Duración / 1000];
            doble angleIncrement = frecuencia / samplingRate;
            doble ángulo = 0;           // normalizada de 0 a 1

            para ( En t i = 0; i <shortBuffer.Length; i++)
            {
                // Definir onda triangular
                doble muestra;

                // 0 a 1
                Si (Ángulo <0.25)
                    muestra = 4 * ángulo;

                // 1--1
                else if (Ángulo <0.75)
                    muestra = 4 * (0.5 - ángulo);

                // -1-0
                más
                    muestra = 4 * (ángulo - 1);

                shortBuffer [i] = ( corto ) (32,767 * muestra);
                ángulo de + = angleIncrement;
            }

            mientras (Ángulo de> 1)
                ángulo - = 1;
        }

        byte [] ByteBuffer = nuevo byte [2 * shortBuffer.Length];
        Buffer .BlockCopy (shortBuffer, 0, ByteBuffer, 0, ByteBuffer.Length);

        DependencyService .get < IPlatformSoundPlayer .> () PlaySound (samplingRate, ByteBuffer);
    }
}
```

3

Aunque las muestras son números enteros de 16 bits, dos de las plataformas que los datos en forma de una matriz de bytes, por lo que se produce una conversión cerca del final con Buffer.BlockCopy. La última línea del método usa DependencyService para pasar esta matriz de bytes con la tasa de muestreo a las plataformas individuales.

los DependencyService.Get referencias de métodos las IPlatformSoundPlayer interfaz que define la firma de la Reproducir sonido método:

```
espacio de nombres MonkeyTapWithSound

{
    public interface IPlatformSoundPlayer
    {
        void Reproducir sonido(En tase de muestreo, byte[] PcmData);
    }
}
```

Ahora viene la parte difícil: escribir este Reproducir sonido Método para las tres plataformas.

Los usos versión de IOS AVAudioPlayer, lo que requiere de datos que incluye la cabecera se utiliza en forma de onda de formato de archivo de audio (.wav). El código aquí que ensambla los datos en una `MemoryBuffer` y luego convierte que a una `NSData` objeto:

utilizando Sistema;
utilizando System.IO;
utilizando System.Text;
utilizando Xamarin.Forms
utilizando AVFoundation;
utilizando Fundación;

[montaje : Dependencia (tipo de (MonkeyTapWithSound.iOS, PlatformSoundPlay

```

writer.Write (16);
writer.Write (( corto ) 1);
writer.Write (( corto ) numChannels);
writer.Write (samplingRate);
writer.Write (samplingRate * numChannels * BitsPerSample / 8); // tasa de bytes
writer.Write (( corto ) (NumChannels * BitsPerSample / 8)); // bloque align
writer.Write (( corto ) BitsPerSample);
writer.Write (nuevo car [] { '\r', '\n', '\T', '\n' }); // fragmento de datos
writer.Write (numSamples * numChannels * BitsPerSample / 8);

// Escribir datos también.

writer.Write (pcmData, 0, pcmData.Length);

memoryStream.Seek (0, SeekOrigin .Empezar);
NSData data = NSData .FromStream (MemoryStream);
AVAudioPlayer audioplayer = AVAudioPlayer .FromData (datos);
audioPlayer.Play ();
}

}

```

Observe los dos elementos esenciales: PlatformSoundPlayer implementa el IPlatformSoundPlayer de la interfaz, y la clase se encuentra en posición con el Dependencia atributo.

La versión de Android utiliza el Pista de audio clase, y que resulta ser un poco más fácil. Sin embargo,

Pista de audio objetos no se pueden superponer, por lo que es necesario para salvar el objeto anterior y detenerlo antes de empezar a jugar el siguiente:

```
utilizando Sistema;
utilizando Android.Media;
utilizando Xamarin.Forms;

[ montaje : Dependencia ( tipo de (MonkeyTapWithSound.Droid. PlatformSoundPlayer ))]

espacio de nombres MonkeyTapWithSound.Droid
{
    clase pública PlatformSoundPlayer : IPlatformSoundPlayer
    {
        Pista de audio previousAudioTrack;

        public void Reproducir sonido( En tasa de muestreo, byte [] PcmData)
        {
            Si (PreviousAudioTrack! = nulo )
            {
                previousAudioTrack.Stop ();
                previousAudioTrack.Release ();
            }
        }

        Pista de audio Audiotrack = nuevo Pista de audio ( Corriente .Múltiples );
        Audiotrack.setVolume (0.5f);
        Audiotrack.setLooping (true);
        Audiotrack.setFormat (PcmDataFormat);
        Audiotrack.setSampleRate (tasa de muestreo);
        Audiotrack.setChannelConfiguration (channelConfiguration);
        Audiotrack.setEncodingFormat (encodingFormat);
        Audiotrack.write (PcmData, 0, PcmData.length);
        Audiotrack.start ();
    }
}
```

[AudioTrackMode](#) .[Estático](#))

Las tres plataformas Windows y Windows Phone pueden utilizar MediaStreamSource. Para evitar una gran cantidad de código repetitivo, la [MonkeyTapWithSound](#) solución contiene un proyecto de SAP adicional denominado

WinRuntimeShared que consiste únicamente de una clase que las tres plataformas se pueden utilizar:

```
utilizando Sistema;  
utilizando System.Runtime.InteropServices.WindowsRuntime;  
utilizando Windows.Media.Core;  
utilizando Windows.Media.MediaProperties;  
utilizando Windows.Storage.Streams;  
utilizando Windows.UI.Xaml.Controls;
```

espacio de nombres MonkeyTapWithSound.WinRuntimeShared

```

    {
        clase pública SharedSoundPlayer
    }

    MediaElement MediaElement = nuevo MediaElement ();
    Espacio de tiempo duración;

    public void Reproducir sonido( En t tasa de muestreo, byte [] PcmData)
    {
        AudioEncodingProperties audioProps =
            AudioEncodingProperties .CreatePcm (( uint ) SamplingRate, 1, 16);
        AudioStreamDescriptor audioDesc = nuevo AudioStreamDescriptor (audioProps);
        MediaStreamSource mss = nuevo MediaStreamSource (audioDesc);

        bool samplePlayed = falso ;
        mss.SampleRequested += (remitente, args) =>
        {
            Si (SamplePlayed)
                regreso ;

            IBuffer ibuffer = pcmData.AsBuffer ();
            MediaStreamSample muestra =
                MediaStreamSample .CreateFromBuffer (ibuffer, Espacio de tiempo . . .
            muestra.Duration = Espacio de tiempo .FromSeconds (pcmData.Length / 2, 0);
            args.Request.Sample = muestra;
            samplePlayed = cierto ;
        };
    }

    mediaElement.SetMediaStreamSource (MSS);
}
}

```

Este proyecto SAP hace referencia a los tres proyectos de Windows Phone y Windows, cada una de las cuales contiene un idéntico (excepto para el espacio de nombres) **PlatformSoundPlayer** clase:

```
utilizando Sistema;
utilizando Xamarin.Forms;

[ montaje : Dependencia ( tipo de (MonkeyTapWithSound.UWP. PlatformSoundPlayer ))]

espacio de nombres MonkeyTapWithSound.UWP
{
    clase pública PlatformSoundPlayer : IPlatformSoundPlayer
    {
        WinRuntimeShared. SharedSoundPlayer sharedSoundPlayer;

        public void Reproducir sonido( En t tasa de muestreo, byte [] PcmData)
        {
            Si (SharedSoundPlayer == nulo )
            {
                sharedSoundPlayer = nuevo WinRuntimeShared. SharedSoundPlayer ();
            }

            sharedSoundPlayer.PlaySound (samplingRate, pcmData);
        }
    }
}
```

El uso de **DependencyService** para llevar a cabo tareas específicas de la plataforma es muy potente, pero este enfoque se queda corto cuando se trata de elementos de interfaz de usuario. Si necesita ampliar el arsenal de puntos de vista que adornan las páginas de sus aplicaciones Xamarin.Forms, ese trabajo implica la creación de procesadores específicos de la plataforma, un proceso discutido en el capítulo final de este libro.

Capítulo 10

extensiones de marcado XAML

En el código, se puede establecer una propiedad en una variedad de maneras diferentes de una variedad de diferentes fuentes:

```
triangle.Angle1 = 45;  
triangle.Angle1 = 180 * radians / Math.PI;  
triangle.Angle1 = ángulos [i];  
triangle.Angle1 = animator.GetCurrentAngle();
```

Si esto angle1 propiedad es una doble, todo lo que se requiere es que la fuente sea una doble o de otra manera proporcionar un valor numérico que es convertible a una doble.

En el marcado, sin embargo, una propiedad de tipo doble por lo general sólo puede ir acompañada de una cadena que califica como un argumento válido para Double.Parse. La única excepción que he visto hasta ahora es cuando la propiedad de destino se marca con una TypeConverter atributo, como la Tamaño de fuente propiedad.

Podría ser deseable si XAML fueran más flexible si se pudiera establecer una propiedad de otras fuentes distintas cadenas de texto explícitos. Por ejemplo, supongamos que desea definir otra manera de establecer una propiedad de tipo Color, tal vez usando el Matiz, Saturación, y Luminosidad Los valores pero sin la molestia de la x: FactoryMethod elemento. Sólo la ligera, no parece posible. El analizador XAML espera que cualquier valor ajustado a un atributo de tipo Color es una cadena aceptable para el ColorTypeConverter clase.

El propósito de XAML *extensiones de marcado* es conseguir alrededor de esta aparente restricción. Tenga la seguridad de que son extensiones de marcado XAML *no* extensiones a XML. XAML es siempre XML legal. extensiones de marcado XAML son extensiones sólo en el sentido de que se extienden las posibilidades de configuración de atributos en el marcado. Una extensión de marcado esencialmente *proporciona* un valor de un tipo particular sin ser necesariamente una representación de texto *de* un valor.

La infraestructura de código

En sentido estricto, una extensión de marcado XAML es una clase que implementa IMarkupExtension, que es una interfaz pública definida en el regulares **Xamarin.Forms.Core** el montaje, pero con el espacio de nombres **Xamarin.Forms.Xaml**:

```
IMarkupExtension interfaz pública  
{  
    oponerse ProvideValue(IServiceProvider ServiceProvider);  
}
```

Como el nombre sugiere, ProvideValue es el método que proporciona un valor a un atributo XAML.

IServiceProvider es parte de las bibliotecas de clases de bases de NET y se define en el Sistema espacio de nombres:

```
interfaz pública IServiceProvider
{
    oponerse GetService (tipo Type);
}
```

Obviamente, esta información no proporciona mucha de una pista sobre cómo escribir extensiones personalizado marcado, y en verdad, que puede ser complicado. (Usted verá un ejemplo breve y otros ejemplos más adelante en este libro.) Afortunadamente, Xamarin.Forms proporciona varias extensiones de marcado valiosa para usted. Estos se dividen en tres categorías:

- extensiones de marcado que son parte de la especificación XAML 2009. Estos aparecen en los archivos XAML con el habitual X prefijo y son:
 - x: Estático
 - x: Referencia
 - x: Tipo
 - x: Null
 - x: Array

Estos se implementan en las clases que consisten en el nombre de la extensión de marcado con la palabra Extensión anexa-por ejemplo, la **StaticExtension** y **ReferenceExtension** clases. Estas clases se definen en el **Xamarin.Forms.Xaml** montaje.

- Las siguientes extensiones de marcado se originaron en el Windows Presentation Foundation (WPF) y, con la excepción de **DynamicResource**, son compatibles con otras implementaciones de Microsoft de XAML, incluyendo Silverlight, Windows Phone 7 y 8 y Windows 8 y 10:
 - StaticResource
 - DynamicResource
 - Unión

Estos se implementan en el público **StaticResourceExtension**, **DynamicResourceExtension**, y **BindingExtension** clases.

- Sólo hay una extensión de marcado que es único para Xamarin.Forms: la **ConstraintExpression** clase usado en conexión con Disposición relativa.

Aunque es posible jugar con las clases de marcas de extensión públicos de código, que en realidad sólo tienen sentido en XAML.

El acceso a miembros estáticos

Una de las implementaciones más simples y útiles de `IMarkupExtension` se encapsula en el `StaticExtension` clase. Esto es parte de la especificación XAML original, por lo que habitualmente aparece en XAML con una X prefijo. `StaticExtension` define una sola propiedad llamada **Miembro** de tipo **cuerda** que se establece a un nombre de clase y miembro de una constante pública, propiedad estática, campo estático, o miembro de la enumeración.

Vamos a ver cómo funciona esto. Aquí está un Etiqueta con seis propiedades determinan como aparecerían normalmente en XAML.

```
< Etiqueta Texto = "Sólo un poco de texto "
    Color de fondo = "Acento "
    Color de texto = "Negro "
    FontAttributes = "Itálico "
    VerticalOptions = "Centrar "
    HorizontalTextAlignment = "Centrar" />
```

Cinco de estos atributos se establecen en cadenas de texto que eventualmente hacen referencia a diversas propiedades estáticas, campos y miembros de la enumeración, pero la conversión de esas cadenas de texto se produce a través de convertidores de tipos y el análisis de XAML nivel de tipos de enumeración.

Si quieres ser más explícito en el establecimiento de estos atributos para esos diversas propiedades, campos y miembros de la enumeración estáticos, puedes utilizar `x: StaticExtension` dentro de las etiquetas de elemento de propiedad:

```
< Etiqueta Texto = "Sólo un poco de texto " >
    < Label.BackgroundColor >
        < x: StaticExtension Miembro = "Color.Accent" />
    </ Label.BackgroundColor >

    < Label.TextColor >
        < x: StaticExtension Miembro = "De color negro" />
    </ Label.TextColor >

    < Label.FontAttributes >
        < x: StaticExtension Miembro = "FontAttributes.Italic" />
    </ Label.FontAttributes >

    < Label.VerticalOptions >
        < x: StaticExtension Miembro = "LayoutOptions.Center" />
    </ Label.VerticalOptions >

    < Label.HorizontalTextAlignment >
        < x: StaticExtension Miembro = "TextAlignment.Center" />
    </ Label.HorizontalTextAlignment >
</ Etiqueta >
```

`Color.Accent` es una propiedad estática. De `color negro` y `LayoutOptions.Center` son campos estáticos.

`FontAttributes.Italic` y `TextAlignment.Center` son miembros de la enumeración.

Teniendo en cuenta la facilidad con que estos atributos se establecen con cadenas de texto, utilizando el enfoque StaticExtension. Inicialmente parece ridícula, pero se dio cuenta que se trata de un mecanismo de propósito general. Puedes usar *alguna* Propiedad estática, campo, o miembro de la enumeración en la StaticExtension etiquetar si su tipo coincide con el tipo de la propiedad de destino.

Por convención, las clases que implementan IMarkupExtension incorporar la palabra Extensión en sus nombres, pero puede dejar que en XAML, por lo que esta extensión de marcado generalmente se llama x: Estático más bien que x: StaticExtension. El siguiente código es ligeramente más corto que el bloque anterior:

```
< Etiqueta Texto = " Sólo un poco de texto " >
    < Label.BackgroundColor >
        < x: Estático Miembro = " Color.Accent " />
    </ Label.BackgroundColor >

    < Label.TextColor >
        < x: Estático Miembro = " De color negro " />
    </ Label.TextColor >

    < Label.FontAttributes >
        < x: Estático Miembro = " FontAttributes.Italic " />
    </ Label.FontAttributes >

    < Label.VerticalOptions >
        < x: Estático Miembro = " LayoutOptions.Center " />
    </ Label.VerticalOptions >

    < Label.HorizontalTextAlignment >
        < x: Estático Miembro = " TextAlignment.Center " />
    </ Label.HorizontalTextAlignment >
</ Etiqueta >
```

Y ahora para el realmente importante reducción del efecto de un marcado cambio en la sintaxis que hace que las etiquetas de propiedad de elementos que desaparecen y la huella para reducir considerablemente. extensiones de marcado XAML casi siempre aparecen con el nombre de extensión de marcado y los argumentos dentro de un par de llaves:

```
< Etiqueta Texto = " Sólo un poco de texto "
    Color de fondo = "{X: miembro estático = Color.Accent} "
    Color de texto = "{X: miembro estático = Color.Black} "
    FontAttributes = "{X: miembro estático = FontAttributes.Italic} "
    VerticalOptions = "{X: miembro estático = LayoutOptions.Center} "
    HorizontalTextAlignment = "{X: miembro estático = TextAlignment.Center} " />
```

Esta sintaxis con las llaves que se utiliza de manera ubicua en conexión con extensiones de marcado XAML que muchos desarrolladores consideran extensiones de marcado a ser sinónimo de la sintaxis rizado corsé. Y eso es casi cierto: mientras que las llaves siempre indican la presencia de una extensión de marcado XAML, en muchos casos, una extensión de marcado puede aparecer en XAML sin las llaves (como se ha demostrado anteriormente) y es a veces conveniente para usarlos de esa manera.

Observe que no hay comillas dentro de las llaves. Dentro de esos apoyos, se aplican diferentes reglas de sintaxis. Los Miembro propiedad de la StaticExtension de clases ya no es un atributo XML. En términos de XML, la expresión completa delimitada por las llaves que es el valor del atributo, y los argumentos dentro de las llaves aparecen sin comillas.

Al igual que los elementos, extensiones de marcado pueden tener una ContentProperty atributo. extensiones de marcado que tienen sólo una propiedad, tales como el StaticExtension clase con su sola Miembro propiedad-marcas que invariablemente propiedad exclusiva como la propiedad de contenido. Para extensiones de marcado utilizando la sintaxis curlybrace, esto significa que el Miembro nombre de la propiedad y el signo igual se pueden quitar:

```
<Etiqueta Texto = "Sólo un poco de texto"
    Color de fondo = "{X: Estático Color.Accent}"
    Color de texto = "{X: Estático Color.Black}"
    FontAttributes = "{X: Estático FontAttributes.Italic}"
    VerticalOptions = "{X: Estático LayoutOptions.Center}"
    HorizontalTextAlignment = "{X: Estático TextAlignment.Center}" />
```

Esta es la forma común de la x: Estático extensión de marcado.

Obviamente, el uso de x: Estático para estas propiedades particulares no es necesaria, pero puede definir sus propios miembros estáticos de la aplicación de las constantes en toda la aplicación, y se puede hacer referencia a estos en sus archivos XAML. Esto se demuestra en el SharedStatics proyecto.

los SharedStatics proyecto contiene una clase llamada AppConstants que define algunas constantes y campos estáticos que pueden ser de uso para el formato de texto:

```
espacio de nombres SharedStatics
{
    clase estática AppConstants
    {
        public static Color LightBackground = Color .Amarillo;
        public static Color DarkForeground = Color .Azul;

        public static double NormalFontSize = 18;
        public static double TitleFontSize = 1,4 * NormalFontSize;
        public static double ParagraphSpacing = 10;

        public const FontAttributes El énfasis = FontAttributes .Itálico;
        public const FontAttributes TitleAttribute = FontAttributes .Negrita;

        public const Alineación del texto TitleAlignment = Alineación del texto .Centrar;
    }
}
```

Se podría utilizar Device.OnPlatform en estas definiciones si necesita algo diferente para cada plataforma.

El archivo XAML continuación, utiliza 18 x: Estático extensiones de marcado para hacer referencia a estos elementos. Note la declaración de espacio de nombres XML que asocia el local prefijo con el espacio de nombres del proyecto:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
```

```
xmlns: x = "http://schemas.microsoft.com/winfx/2009/xaml"
xmlns: locales = "CLR-espacio de nombres: SharedStatics "
x: Class = "SharedStatics.SharedStaticsPage "
Color de fondo = "(X: local de estática: AppConstants.LightBackground)" >

< ContentPage.Padding >
    < OnPlatform x: TypeArguments = "Espesor "
        iOS = "0, 20, 0, 0" />
</ ContentPage.Padding >

< StackLayout Relleno = "10, 0 "
    Espaciado = "(X: local de estática: AppConstants.ParagraphSpacing)" >

    < Etiqueta Texto = "El Programa SharedStatics "
        Color de texto = "(X: local de estática: AppConstants.DarkForeground)"
        Tamaño de fuente = "(X: local de estática: AppConstants.TitleFontSize)"
        FontAttributes = "(X: local de estática: AppConstants.TitleAttribute)"
        HorizontalTextAlignment = "(X: local de estática: AppConstants.TitleAlignment)" />

    < Etiqueta Color de texto = "(X: local de estática: AppConstants.DarkForeground)"
        Tamaño de fuente = "(X: local de estática: AppConstants.NormalFontSize)" >
        < Label.FormattedText >
            < FormattedString >
                < Lapso Texto = "A través del uso de la " />
                < Lapso Texto = "x: Estático "
                    Tamaño de fuente = "(X: local de estática: AppConstants.NormalFontSize)"
                    FontAttributes = "(X: local de estática: AppConstants.Emphasis)" />
                < Lapso Texto =
                    " extensión de marcado XAML, una aplicación puede mantener una colección de
                    configuración de las propiedades comunes definidos como constantes, propiedades estáticas o campos,
                    o miembros de la enumeración en un archivo de código separado. Estos pueden ser
                    hace referencia el archivo XAML." />
                    </ FormattedString >
                </ Label.FormattedText >
            </ Etiqueta >

            < Etiqueta Color de texto = "(X: local de estática: AppConstants.DarkForeground)"
                Tamaño de fuente = "(X: local de estática: AppConstants.NormalFontSize)" >
                < Label.FormattedText >
                    < FormattedString >
                        < Lapso Texto =
                            " Sin embargo, esta no es la única técnica para compartir la configuración de propiedades.
                            Pronto descubrirá que puede almacenar objetos en una " />
                            < Lapso Texto = " ResourceDictionary "
                                Tamaño de fuente = "(X: local de estática: AppConstants.NormalFontSize)"
                                FontAttributes = "(X: local de estática: AppConstants.Emphasis)" />
                            < Lapso Texto = " y acceder a ellos a través de la " />
                            < Lapso Texto = " StaticResource "
                                Tamaño de fuente = "(X: local de estática: AppConstants.NormalFontSize)"
                                FontAttributes = "(X: local de estática: AppConstants.Emphasis)" />
                            < Lapso Texto =
                                " marcado extensión, e incluso encapsulate múltiples configuraciones de propiedad en una " />
                                < Lapso Texto = " Estilo "
                                    Tamaño de fuente = "(X: local de estática: AppConstants.NormalFontSize)"
```

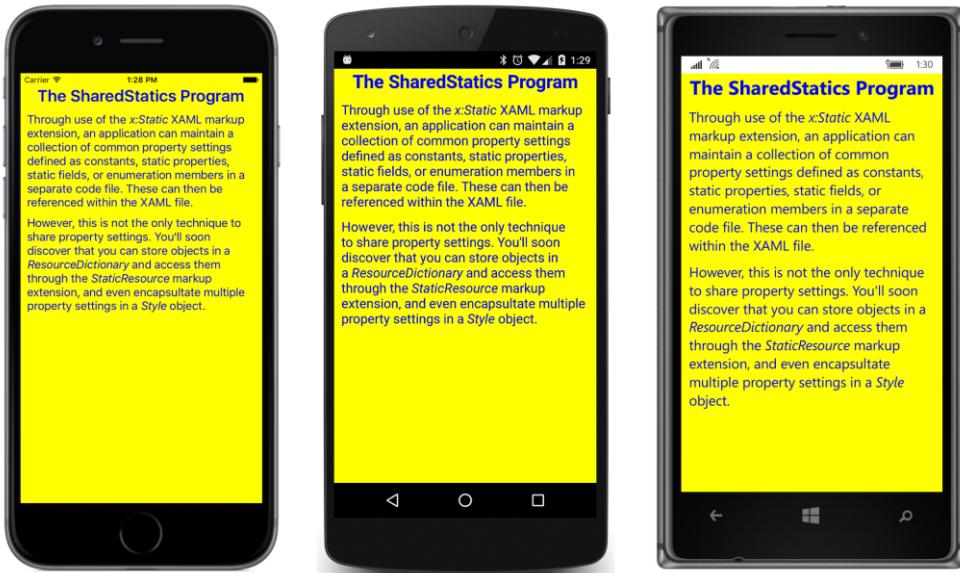
```

        FontAttributes = "X: local de estática: AppConstants.Emphasis" />
    <Lapso Texto = "objeto." />
</FormattedString>
<Label.FormattedText>
</Etiqueta>
</StackLayout>
</Pagina de contenido>

```

Cada una de las Lapso objetos con una FontAttributes repite el entorno Tamaño de fuente configuración que se establece en el Etiqueta sí, porque Lapso objetos no heredan los ajustes relacionados con las fuentes de la Etiqueta cuando se aplica otra configuración relacionada con las fuentes.

Y aquí está:



Esta técnica permite el uso de estos valores de propiedades comunes en varias páginas, y si alguna vez tiene que cambiar los valores, sólo es necesario cambiar el Ajustes de Aplicacion archivo.

También es posible utilizar x: Estático con propiedades estáticas y campos definidos en las clases en las bibliotecas externas. En el siguiente ejemplo, el nombre **SystemStatics**, es más bien artificial, que establece el Ancho del borde de un Botón igual a la Pi campo estático se define en la Mates clase y utiliza la estática Environment.NewLine viviendo en los saltos de línea en el texto. Pero demuestra la técnica.

los Mates y Ambiente las clases se definen tanto en el .NET Sistema espacio de nombres, por lo que se requiere una nueva declaración de espacio de nombres XML para definir un prefijo de llamada (por ejemplo) sys para Sistema. Tenga en cuenta que esta declaración de espacio de nombres especifica el espacio de nombres CLR Sistema pero el conjunto como se msclorib, que se situó originalmente para Microsoft Common Object Runtime Library pero ahora representa multilenguaje estándar Common Object Runtime Library:

```

< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    xmlns: sys = "clr-espacio de nombres: System; montaje = mscorelib "
    x: Class = "SystemStatics.SystemStaticsPage " >

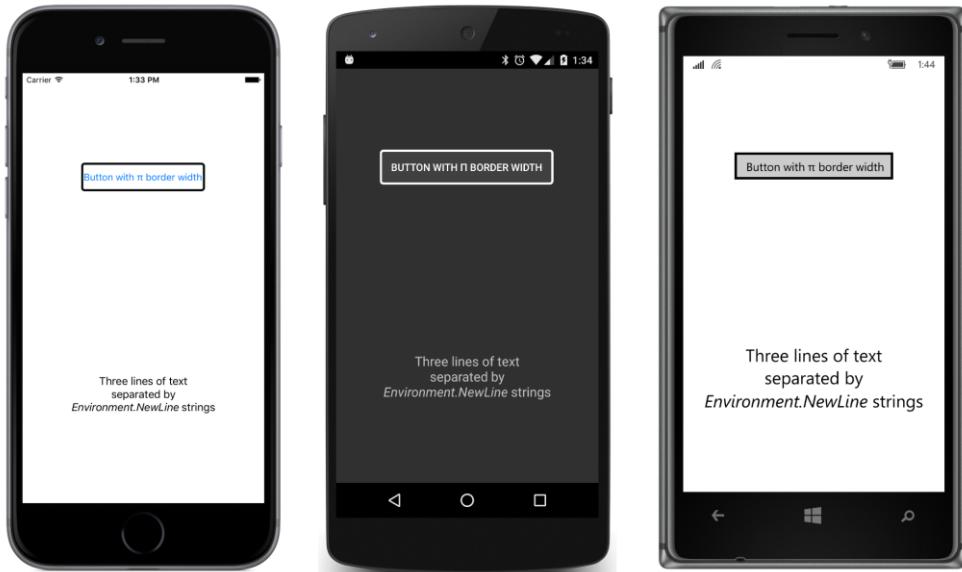
    < StackLayout >
        < Botón Texto = " botón con & # X03C0; ancho del borde "
            Ancho del borde = " {x: Estático sys: Math.PI} "
            HorizontalOptions = " Centrar "
            VerticalOptions = " CenterAndExpand " >
            < Button.BackgroundColor >
                < OnPlatform x: TypeArguments = " Color "
                    Androide = "# 404040 " />
            </ Button.BackgroundColor >
            < Button.BorderColor >
                < OnPlatform x: TypeArguments = " Color "
                    Androide = " Blanco "
                    WinPhone = " Negro " />
            </ Button.BorderColor >
        </ Botón >

        < Etiqueta VerticalOptions = " CenterAndExpand "
            HorizontalTextAlignment = " Centrar "
            Tamaño de fuente = " Medio " >
            < Label.FormattedText >
                < FormattedString >
                    < Lazo Texto = " Tres líneas de texto " />
                    < Lazo Texto = " (X: sys estáticas: Environment.NewLine) " />
                    < Lazo Texto = " separado por " />
                    < Lazo Texto = " (X: sys estáticas: Environment.NewLine) " />
                    < Lazo Texto = " Environment.NewLine "
                        Tamaño de fuente = " Medio "
                        FontAttributes = " Itálico " />
                    < Lazo Texto = " instrumentos de cuerda " />
                </ FormattedString >
            </ Label.FormattedText >
        </ Etiqueta >
    </ StackLayout >
</ Pagina de contenido >

```

El borde del botón no aparece en Android a menos que se establece el color de fondo, y tanto en Android y Windows Phone necesita la frontera de un color no predeterminada, por lo que algunos métodos de marcado se hace cargo de esos problemas. En las plataformas iOS, una frontera botón tiende a desplazar el texto del botón, por lo que el texto se define con espacios al principio y al final.

A juzgar únicamente de los elementos visuales, que realmente tiene que tomar en la confianza de que la anchura botón frontera es de aproximadamente 3,14 unidades de ancho, pero la línea se rompe definitivamente el trabajo:



El uso de llaves para extensiones de marcado implica que no se puede mostrar el texto rodeado por llaves. Las llaves en este texto se pueden confundir con una extensión de marcado:

```
<Etiqueta Texto = " {Texto entre llaves} " />
```

Eso no funcionará. Usted puede tener las llaves en la cadena de texto en otro lugar, pero no se puede comenzar con una llave izquierda.

Si realmente necesita, sin embargo, puede asegurarse de que el texto no es confundido con una extensión de marcado XAML al comenzar el texto con una secuencia de escape que consiste en un par de llaves izquierdo y derecho:

```
<Etiqueta Texto = " {{Texto entre llaves}} " />
```

Esto debe mostrar el texto que deseé.

diccionarios de recursos

Xamarin.Forms también soporta un segundo enfoque para compartir objetos y valores, y mientras este enfoque tiene un poco más de sobrecarga que la x: Estático extensión de marcado, es un poco más versátil, porque todo: los objetos compartidos y los elementos visuales que utilizan ellos pueden expresarse en XAML.

VisualElement define una propiedad denominada recursos que es de tipo ResourceDictionary -un diccionario con cuerdas claves y valores de tipo objeto. Los productos que se pueden añadir a este diccionario justo en XAML, y se puede acceder en XAML con el StaticResource y DynamicResource extensiones de marcado.

A pesar de que x: Estático y StaticResource tienen nombres un tanto similares, que son muy diferentes:

x: Estático hace referencia a una, un campo constante estática, una propiedad estática, o un miembro de la enumeración, mientras StaticResource recupera un objeto de una ResourceDictionary.

Mientras que la x: Estático extensión de marcado es intrínseca a XAML (y por tanto aparece en XAML con una X prefijo), la StaticResource y DynamicResource extensiones de marcado no lo son. Eran parte de la aplicación XAML original en Windows Presentation Foundation, y StaticResource también es compatible con Silverlight, Windows Phone 7 y 8 y Windows 8 y 10.

Vamos a usar StaticResource para la mayoría de propósitos y reserva DynamicResource para algunas aplicaciones especiales, por lo que vamos a comenzar con StaticResource.

StaticResource para la mayoría de los propósitos

Supongamos que hemos definido tres botones en una StackLayout:

```
< StackLayout >
    < Botón Texto = " Carpe Diem "
        HorizontalOptions = " Centrar "
        VerticalOptions = " CenterAndExpand "
        Ancho del borde = " 3 "
        Color de texto = " rojo "
        Tamaño de fuente = " Grande " >
        < Button.BackgroundColor >
            < OnPlatform x: TypeArguments = " Color "
                Androide = "# 404040 " />
        </ Button.BackgroundColor >
        < Button.BorderColor >
            < OnPlatform x: TypeArguments = " Color "
                Androide = " Blanco "
                WinPhone = " Negro " />
        </ Button.BorderColor >
    </ Botón >

    < Botón Texto = " Sapere aude "
        HorizontalOptions = " Centrar "
        VerticalOptions = " CenterAndExpand "
        Ancho del borde = " 3 "
        Color de texto = " rojo "
        Tamaño de fuente = " Grande " >
        < Button.BackgroundColor >
            < OnPlatform x: TypeArguments = " Color "
                Androide = "# 404040 " />
        </ Button.BackgroundColor >
        < Button.BorderColor >
            < OnPlatform x: TypeArguments = " Color "
                Androide = " Blanco "
                WinPhone = " Negro " />
        </ Button.BorderColor >
    </ Botón >

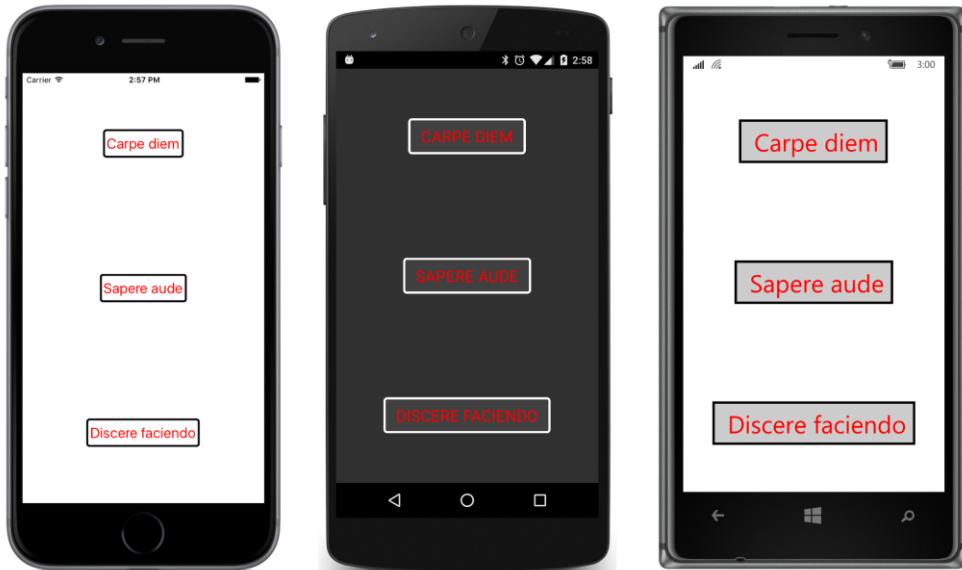
    < Botón Texto = " haciendo discore "
        HorizontalOptions = " Centrar "
        VerticalOptions = " CenterAndExpand "
        Ancho del borde = " 3 "
        Color de texto = " rojo "
        Tamaño de fuente = " Grande " >
        < Button.BackgroundColor >
            < OnPlatform x: TypeArguments = " Color "
                Androide = "# 404040 " />
        </ Button.BackgroundColor >
        < Button.BorderColor >
            < OnPlatform x: TypeArguments = " Color "
                Androide = " Blanco "
                WinPhone = " Negro " />
        </ Button.BorderColor >
    </ Botón >
```

```

    HorizontalOptions = "Centrar"
    VerticalOptions = "CenterAndExpand"
    Ancho del borde = "3"
    Color de texto = "rojo"
    Tamaño de fuente = "Grande"
< Button.BackgroundColor >
    < OnPlatform x: TypeArguments = "Color"
        Androide = "# 404040" />
</ Button.BackgroundColor >
< Button.BorderColor >
    < OnPlatform x: TypeArguments = "Color"
        Androide = "Blanco"
        WinPhone = "Negro" />
</ Button.BorderColor >
</ Botón >
</ StackLayout >

```

Por supuesto, esto es poco realista. No existen hecho clic eventos fijados para estos botones, y en general el texto del botón no es en América. Pero aquí está lo que parecen:



Aparte del texto, los tres botones tienen las mismas propiedades configuradas con los mismos valores. Repetitivo marcado como esto tiende a frotar los programadores de la manera incorrecta. Es una afrenta a la vista y de difícil mantenimiento y cambio.

Con el tiempo verás cómo utilizar estilos de cortar muy abajo en el marcado repetitivo. Por ahora, sin embargo, el objetivo no es hacer que el margen de beneficio más corto, pero para consolidar los valores en un solo lugar por lo que si alguna vez desea cambiar el Color de texto propiedad de rojo a Azul, puede hacerlo con una sola edición en lugar de tres.

Obviamente, se puede utilizar x: Estático para este trabajo mediante la definición de los valores en el código. Pero vamos a hacer todo el asunto en XAML mediante el almacenamiento de los valores de una *diccionario de recursos*. Cada clase que deriva de `VisualElement` tiene un recurso propiedad de tipo `ResourceDictionary`. Los recursos que se utilizan en una página se almacenan normalmente en la recursos colección de la `Page` de contenido.

El primer paso es para expresar la recursos propiedad de `Page` de contenido como un elemento de propiedad:

```
< Page de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " ResourceSharing.ResourceSharingPage " >

    < ContentPage.Resources >
        ...
    </ ContentPage.Resources >
</ Page de contenido >
```

Si también está definiendo una `Relleno` propiedad en la página mediante el uso de etiquetas de propiedad de elementos, el orden no es importante.

Por motivos de rendimiento, la recursos propiedad es nulo de forma predeterminada, por lo que necesita para crear una instancia explícitamente la `ResourceDictionary`:

```
< Page de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " ResourceSharing.ResourceSharingPage " >

    < ContentPage.Resources >
        < ResourceDictionary >
            ...
        </ ResourceDictionary >
    </ ContentPage.Resources >
</ Page de contenido >
```

Entre los `ResourceDictionary` etiquetas, se definen uno o más objetos o valores. Cada elemento en el diccionario debe ser identificado con una clave de diccionario que se especifica con el XAML `x: Key` atributo. Por ejemplo, aquí es la sintaxis para la inclusión de una `LayoutOptions` valor en el diccionario con una clave descriptivo que indica que este valor se define para configurar las opciones horizontales:

```
< LayoutOptions x: Key = " horzOptions " > Centrar </ LayoutOptions >
```

Debido a que esta es una `LayoutOptions` valor, el analizador XAML accede a la `LayoutOptionsConverter` clase para convertir el contenido de las etiquetas, que es el texto “centro”.

Una segunda manera de almacenar una `LayoutOptions` valor en el diccionario es dejar que el analizador XAML instanciar la estructura y establecer `LayoutOptions` propiedades de los atributos que especifique:

```
< LayoutOptions x: Key = " vertOptions "
    Alineación = " Centrar "
    expande = " Ciento " />
```

los Ancho del borde propiedad es de tipo doble, entonces el x: Doble elemento de tipo de datos definido en la especificación XAML 2009 es ideal:

```
<x: Doble x: Key = "ancho del borde" > 3 </x: Doble >
```

Puede almacenar una Color valor en el diccionario de recursos con una representación de texto del color que el contenido. El analizador XAML utiliza la normalidad ColorTypeConverter para la conversión de texto:

```
<Color x: Key = "color de texto" > rojo </Color >
```

También puede especificar los valores hexadecimales ARGB continuación de una almohadilla.

No se puede inicializar una Color valor estableciendo su R, GRAMO, y segundo porque esas son las propiedades de sólo conseguir. Pero se puede invocar una Color constructor usando x: Argumentos o una de las Color métodos de fábrica utilizando x: FactoryMethod y x: argumentos.

```
<Color x: Key = "color de texto"
      x: FactoryMethod = "FromHsla" >
<x: Argumentos>
  <x: Doble> 0 </x: Doble>
  <x: Doble> 1 </x: Doble>
  <x: Doble> 0.5 </x: Doble>
  <x: Doble> 1 </x: Doble>
</x: Argumentos>
</Color >
```

Notar tanto la x: Key y x: FactoryMethod atributos.

los Color de fondo y Color del borde propiedades de los tres botones mostrados anteriormente se establecen en valores de la OnPlatform clase. Afortunadamente se puede poner OnPlatform objetos justo en el diccionario:

```
<OnPlatform x: Key = "color de fondo"
            x: TypeArguments = " Color "
            Androide = "# 404040 " />

<OnPlatform x: Key = "color del borde"
            x: TypeArguments = " Color "
            Androide = "Blanco"
            WinPhone = "Negro" />
```

Notar tanto la x: Key y x: TypeArguments atributos.

Un elemento de diccionario para la Tamaño de fuente la propiedad es un tanto problemática. los Tamaño de fuente propiedad es de tipo doble, por lo que si usted está almacenando un valor numérico real en el diccionario, eso no es problema. Pero no se puede almacenar la palabra "grande" en el diccionario como si se tratara de una doble. Sólo cuando una cadena "grande" está ajustado a una Tamaño de fuente atributo usa el analizador XAML la FontSizeConverter. Por esa razón, es necesario almacenar la Tamaño de fuente elemento como una cadena:

```
<x: String x: Key = "tamaño de fuente" > Grande </x: String >
```

Aquí está el diccionario completo en este punto:

```

< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " ResourceSharing.ResourceSharingPage " >

    < ContentPage.Resources >
        < ResourceDictionary >
            < LayoutOptions x: Key = " horzOptions " > Centrar </ LayoutOptions >

            < LayoutOptions x: Key = " vertOptions " *
                Alineación = " Centrar "
                expande = " Ciento " />

            < x: Doble x: Key = " ancho del borde " > 3 < x: Doble >

            < Color x: Key = " color de texto " > rojo < Color >

            < OnPlatform x: Key = " color de fondo "
                x: TypeArguments = " Color "
                Androide = "#404040" />

            < OnPlatform x: Key = " color del borde "
                x: TypeArguments = " Color "
                Androide = " Blanco "
                WinPhone = " Negro " />

            < x: String x: Key = " tamaño de fuente " > Grande < x: String >
        </ ResourceDictionary >
    </ ContentPage.Resources >
    ...
</ Pagina de contenido >
```

Esto se refiere a veces como una *sección de recursos* para la página. En la programación de la vida real, muchos archivos XAML comienzan con una sección de recursos.

Puede hacer referencia a elementos en el diccionario mediante el uso de la `StaticResource` extensión de marcado, que está soportado por `StaticResourceExtension`. La clase define una propiedad denominada `Llave` que establezca la clave de diccionario. Puede utilizar una `StaticResourceExtension` como un elemento dentro de las etiquetas de propiedad de elementos, o puede utilizar `StaticResourceExtension` o `StaticResource` entre llaves. Si está utilizando la sintaxis rizado corsé, puede dejar de lado el `Llave` y el signo igual, porque `Llave` es la propiedad de contenido `StaticResourceExtension`.

El siguiente archivo XAML completa en el **El intercambio de recursos** proyecto ilustra tres de estas opciones:

```

< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " ResourceSharing.ResourceSharingPage " >

    < ContentPage.Resources >
        < ResourceDictionary >
            < LayoutOptions x: Key = " horzOptions " > Centrar </ LayoutOptions >

            < LayoutOptions x: Key = " vertOptions " *
                Alineación = " Centrar "
```

```
        expande = " Ciento " />

    < x: Doble x: Key = " ancho del borde " > 3 < x: Doble >

    < Color x: Key = " color de texto " > rojo </ Color >

    < OnPlatform x: Key = " color de fondo "
      x: TypeArguments = " Color "
      Androide = "# 404040 " />

    < OnPlatform x: Key = " color del borde "
      x: TypeArguments = " Color "
      Androide = " Blanco "
      WinPhone = " Negro " />

    < x: String x: Key = " tamaño de fuente " > Grande < x: String >
  </ ResourceDictionary >
</ ContentPage.Resources >

< StackLayout >
  < Botón Texto = " Carpe Diem " >
    < Button.HorizontalOptions >
      < StaticResourceExtension Llave = " horzOptions " />
    </ Button.HorizontalOptions >

    < Button.VerticalOptions >
      < StaticResourceExtension Llave = " vertOptions " />
    </ Button.VerticalOptions >

    < Button.BorderWidth >
      < StaticResourceExtension Llave = " ancho del borde " />
    </ Button.BorderWidth >

    < Button.TextColor >
      < StaticResourceExtension Llave = " color de texto " />
    </ Button.TextColor >

    < Button.BackgroundColor >
      < StaticResourceExtension Llave = " color de fondo " />
    </ Button.BackgroundColor >

    < Button.BorderColor >
      < StaticResourceExtension Llave = " color del borde " />
    </ Button.BorderColor >

    < Button.FontSize >
      < StaticResourceExtension Llave = " tamaño de fuente " />
    </ Button.FontSize >
  </ Botón >

  < Botón Texto = " Sapere aude "
    HorizontalOptions = "{StaticResource Clave = horzOptions}"
    VerticalOptions = "{StaticResource Clave = vertOptions}"
    Ancho del borde = "{StaticResource Clave = borderWidth}"
```

```

Color de texto = "{StaticResource Clave = textColor}"
Color de fondo = "{StaticResource Clave = backgroundColor}"
Color del borde = "{StaticResource Clave = borderColor}"
Tamaño de fuente = "{StaticResource Clave = fontSize}" />

< Botón Texto = "faciendo discere"
    HorizontalOptions = "{} StaticResource horzOptions"
    VerticalOptions = "{} StaticResource vertOptions"
    Ancho del borde = "{} StaticResource borderWidth"
    Color de texto = "{} StaticResource textColor"
    Color de fondo = "{} StaticResource backgroundColor"
    Color del borde = "{} StaticResource borderColor"
    Tamaño de fuente = "{} StaticResource fontSize" />

</ StackLayout >
</ Pagina de contenido >

```

La sintaxis más simple en el tercer botón es el más común, y, de hecho, que la sintaxis es tan omnipresente que muchos desarrolladores XAML de toda la vida podrían ser totalmente familiarizado con las otras variantes. Sin embargo, si utiliza una versión de StaticResource con el Llave propiedad, no poner X prefijo en él. los x: Key atributo es sólo para definir las claves de diccionario para los elementos de la ResourceDictionary.

Objetos y valores en el diccionario se comparten entre todos los StaticResource referencias. Eso no es tan claro en el ejemplo anterior, pero es algo a tener en cuenta. Por ejemplo, supongamos que almacena una Botón objeto en el diccionario de recursos:

```

< ContentPage.Resources >
    < ResourceDictionary >
        < Botón x: Key = "botón"
            Texto = "Botón compartido?"
            HorizontalOptions = "Centrar"
            VerticalOptions = "CenterAndExpand"
            Tamaño de fuente = "Grande" />
    </ ResourceDictionary >
< ContentPage.Resources >

```

Por supuesto que puede utilizar ese Botón oponerse en su página añadiéndolo a la Niños cobro de una StackLayout con el StaticResourceExtension la sintaxis de elementos:

```

< StackLayout >
    < StaticResourceExtension Llave = "botón" />
< StackLayout >

```

Sin embargo, no se puede utilizar el mismo elemento de diccionario con la esperanza de poner otra copia en el StackLayout fuera:

```

< StackLayout >
    < StaticResourceExtension Llave = "botón" />
    < StaticResourceExtension Llave = "botón" />
< StackLayout >

```

Eso no funcionará. Ambos elementos de referencia de la misma Botón objeto, y un elemento visual en particular puede estar en un lugar en particular en la pantalla. No puede estar en varios lugares.

Por esta razón, los elementos visuales no se almacenan normalmente en un diccionario de recursos. Si necesita varios elementos de la página que tienen en su mayoría las mismas propiedades, tendrá que utilizar una Estilo, el cual se explora en el capítulo 12.

Un árbol de diccionarios

los ResourceDictionary clase impone las mismas reglas que otros diccionarios: todos los elementos en el diccionario deben tener llaves, pero claves duplicadas no están permitidos.

Sin embargo, debido a que cada instancia de VisualElement potencialmente tiene su propio diccionario de recursos, la página puede contener múltiples diccionarios, y se puede utilizar las mismas claves en diferentes diccionarios con tal de que todas las llaves dentro de cada diccionario son únicos. Es concebible que cada elemento visual en el árbol visual puede tener su propio diccionario, pero en realidad sólo tiene sentido para un diccionario de recursos para aplicar a múltiples elementos, por lo que sólo están diccionarios de recursos que se encuentran comúnmente definida en Diseño o Página

objetos.

Mediante esta técnica se puede construir un árbol de diccionarios con claves de diccionario que anulan efectivamente las llaves en otros diccionarios. Esto se demuestra en el **ResourceTrees** proyecto. El archivo XAML para el ResourceTreesPage clase muestra una recursos Inglés para el Pagina de contenido que define los recursos con las teclas de horzOptions, vertOptions, y color de texto.

Un segundo recursos diccionario está unido a un interior StackLayout de recursos con nombre color de texto y Tamaño de fuente:

```
<Pagina de contenido xmlns = "http://xamarin.com/schemas/2014/forms"
    xmlns:x = "http://schemas.microsoft.com/winfx/2009/xaml"
    x: Class = "ResourceTrees.ResourceTreesPage">

    <ContentPage.Resources>
        <ResourceDictionary>
            <LayoutOptions x: Key = "horzOptions" >Centrar</LayoutOptions>

            <LayoutOptions x: Key = "vertOptions" 
                Alineación = "Centrar"
                expande = "Cielo" />

            <OnPlatform x: Key = "color de texto" 
                x: TypeArguments = "Color"
                iOS = "rojo"
                Androide = "Rosado"
                WinPhone = "Azul" />
        </ResourceDictionary>
    </ContentPage.Resources>

    <StackLayout>
        <Botón Texto = "Carpe Diem" 
            HorizontalOptions = "0 StaticResource horzOptions"
            VerticalOptions = "0 StaticResource vertOptions"
            Ancho del borde = "0 StaticResource borderWidth"
            Color de texto = "0 StaticResource textColor" />
    </StackLayout>

```

```

Color de fondo = " {0} StaticResource backgroundColor "
Color del borde = " {0} StaticResource borderColor "
Tamaño de fuente = " {0} StaticResource fontSize " />

< StackLayout >
  < StackLayout.Resources >
    < ResourceDictionary >
      < Color x: Key = " color de texto " > Defecto </ Color >
      < x: String x: Key = " tamaño de fuente " > Defecto </ x: String >
    </ ResourceDictionary >
  </ StackLayout.Resources >

  < Etiqueta Texto = " La primera de las dos etiquetas " >
    HorizontalOptions = " {0} StaticResource horzOptions "
    Color de texto = " {0} StaticResource textColor "
    Tamaño de fuente = " {0} StaticResource fontSize " />

  < Botón Texto = " Sapere aude " >
    HorizontalOptions = " {0} StaticResource horzOptions "
    Ancho del borde = " {0} StaticResource borderWidth "
    Color de texto = " {0} StaticResource textColor "
    Color de fondo = " {0} StaticResource backgroundColor "
    Color del borde = " {0} StaticResource borderColor "
    Tamaño de fuente = " {0} StaticResource fontSize " />

  < Etiqueta Texto = " El segundo de dos etiquetas " >
    HorizontalOptions = " {0} StaticResource horzOptions "
    Color de texto = " {0} StaticResource textColor "
    Tamaño de fuente = " {0} StaticResource fontSize " />
</ StackLayout >

< Botón Texto = " faciendo discere " >
  HorizontalOptions = " {0} StaticResource horzOptions "
  VerticalOptions = " {0} StaticResource vertOptions "
  Ancho del borde = " {0} StaticResource borderWidth "
  Color de texto = " {0} StaticResource textColor "
  Color de fondo = " {0} StaticResource backgroundColor "
  Color del borde = " {0} StaticResource borderColor "
  Tamaño de fuente = " {0} StaticResource fontSize " />
</ StackLayout >
</ Pagina de contenido >
```

los recursos diccionario en el interior StackLayout se aplica únicamente a los elementos dentro de ese StackLayout, fuera, los cuales son los elementos en el medio de captura de pantalla:



Así es como funciona:

Cuando el analizador XAML se encuentra con una `StaticResource` en un atributo de un elemento visual, comienza una búsqueda para esa clave de diccionario. Primero busca en el `ResourceDictionary` para ese elemento visual, y si no se encuentra la clave, busca la clave en la matriz de del elemento visual `ResourceDictionary`, y arriba y arriba a través del árbol visual hasta que llega al `ResourceDictionary` en la pagina.

Pero falta algo aquí! ¿Dónde están las entradas en la página de `ResourceDictionary` para `BorderWidth`, color de fondo, color del borde, y ¿tamaño de fuente? No están en el archivo `ResourceTreesPage.xaml`!

Esos artículos están en otra parte. Los `Solicitud` clase de la que de todas las aplicaciones Aplicación clase deriva-también define una recursos propiedad de tipo `ResourceDictionary`. Esto es útil para la definición de los recursos que se aplican a toda la aplicación y no sólo a una página o diseño en particular. Cuando el analizador XAML busca en el árbol visual para una llave recurso coincidente, y esa llave no se encuentra en el `ResourceDictionary` para la página, que finalmente se comprueba el `ResourceDictionary` definida por la `Aplicación` clase. Sólo si no se encuentra allí es una `XamlParseException` planteado para el `StaticResource` error de clave-no-encontrado.

Se pueden añadir elementos a su Aplicación la clase de `ResourceDictionary` objeto de dos maneras:

Un método consiste en añadir los elementos de código en el Aplicación constructor. Asegúrese de hacer esto antes de crear instancias de los principales Pagina de contenido clase:

```
público clase Aplicación : Solicitud
{
    público App ()
    {
    }
```

```

recursos = nuevo ResourceDictionary ();
Resources.Add ("ancho del borde" , 3.0);
Resources.Add ("tamaño de fuente" , "Grande");
Resources.Add ("color de fondo" ,
    Dispositivo .OnPlatform ( Color .Defecto,
        Color .FromArgb (0x40, 0x40, 0x40),
        Color .Defecto));

Resources.Add ("color del borde" ,
    Dispositivo .OnPlatform ( Color .Defecto,
        Color .Blanco,
        Color .Negro));

MainPage = nuevo ResourceTreesPage ();
}
...
}

```

sin embargo, el Aplicación clase también puede tener un archivo XAML de su propia, y los recursos en toda la aplicación puede ser definido en el recursos recogida en ese archivo XAML. Para ello, tendrá que eliminar el archivo App.cs creado por la plantilla de solución Xamarin.Forms. No hay elemento de la plantilla para una Aplicación clase, por lo que tendrá que fingir. Añadir una nueva página de clase **XAML Página forma Xaml en Visual Studio o ContentPage forma Xaml Estudio en Xamarin-al proyecto. Nombralo App. E inmediatamente antes olvida-go en el archivo App.xaml y cambiar las etiquetas de raíz para Solicitud, y entrar en el archivo App.xaml.cs y cambiar la clase base a Solicitud.**

Ahora usted tiene una Aplicación clase que deriva de Solicitud y tiene su propio archivo XAML. En el archivo App.xaml a continuación, puede crear una instancia de una ResourceDictionary dentro Application.Resources etiquetas de propiedad de elementos y añadir elementos a ella:

```

< Solicitud xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " ResourceTrees.App " >

< Application.Resources >
    < ResourceDictionary >
        < x: Doble x: Key = " ancho del borde " > 3 </ x: Doble >
        < x: String x: Key = " tamaño de fuente " > Grande </ x: String >

        < OnPlatform x: Key = " color de fondo " >
            x: TypeArguments = " Color "
            Androide = "# 404040 " />

        < OnPlatform x: Key = " color del borde " >
            x: TypeArguments = " Color "
            Androide = " Blanco "
            WinPhone = " Negro " />
    </ ResourceDictionary >
</ Application.Resources >
</ Solicitud >

```

El constructor en el archivo de código subyacente tiene que llamar InitializeComponent para analizar el App.xaml

presentar en tiempo de ejecución y añadir los elementos al diccionario. Esto debe hacerse antes de que el trabajo normal de crear instancias de la ResourceTreesPage clase y se establece la Pagina principal propiedad:

```
público clase parcial Aplicación : Solicitud
{
    público App ()
    {
        InitializeComponent ();

        MainPage = nuevo ResourceTreesPage ();
    }

    protegido override void OnStart ()
    {
        // manipulador cuando se inicia el app
    }

    protegido override void OnSleep ()
    {
        // manipulador cuando su aplicación duerme
    }

    protegido override void En resumen()
    {
        // manejar cuando sus hojas de vida de aplicaciones
    }
}
```

La adición de los eventos del ciclo de vida es opcional.

Asegúrese de llamar InitializeComponent antes de crear instancias de la clase de página. El constructor de la clase de página llama a su propia InitializeComponent para analizar el archivo XAML para la página, y el Stat

icResource extensiones de marcado necesitan tener acceso a la recursos recogida en el Aplicación clase.

Cada recursos diccionario tiene un ámbito concreto: Para el recursos en el diccionario Aplicación clase, que el alcance es toda la aplicación. UN recursos en el diccionario Pagina de contenido clase corresponde a toda la página. UN recursos en un diccionario StackLayout se aplica a todos los niños en el StackLay- fuera. Debe definir y almacenar los recursos en función de la forma de usarlos. Utilizar el recursos en el diccionario Aplicación clase de recursos en toda la aplicación; utilizar el recursos en el diccionario Estafa- tentPage de recursos de toda la página; pero definir adicional recursos diccionarios más profundas en el árbol visual de los recursos necesarios sólo en una parte de la página.

Como se verá en el capítulo 12, los elementos más importantes en una recursos diccionario suelen ser objetos de tipo Estilo. En el caso general, tendrá aplicación a escala Estilo objetos, Estilo objetos de la página, y Estilo objetos asociados con las piezas más pequeñas del árbol visual.

DynamicResource para fines especiales

Una alternativa a las StaticResource para hacer referencia a elementos de la recursos diccionario es Dynam- icResource, y si usted acaba de sustituir DynamicResource para StaticResource en el ejemplo

mostrado anteriormente, el programa aparentemente ejecutar el mismo. Sin embargo, las dos extensiones de marcado son muy diferentes. StaticResource accede al elemento en el diccionario de una sola vez, mientras que el XAML está siendo analizada y la página está en construcción. Pero DynamicResource mantiene un vínculo entre la clave de diccionario y el conjunto de propiedades de ese elemento del diccionario. Si el artículo en el diccionario de recursos referenciados por los cambios clave, DynamicResource detectará que el cambio y establecer el nuevo valor de la propiedad.

¿Escéptico? Vamos a probarlo. Los **DynamicVsStatic** proyecto tiene un archivo XAML que define un elemento de recurso de tipo cuerda con una clave de CurrentDateTime, a pesar de que el artículo en el diccionario es la cadena "No es en realidad un DateTime"!

Este elemento de diccionario se hace referencia a cuatro veces en el archivo XAML, pero una de las referencias está comentada. En los dos primeros ejemplos, la Texto propiedad de una Etiqueta se ajusta utilizando StaticResource y DynamicResource. En el segundo de dos ejemplos, la Texto propiedad de una Lapso objeto se establece de manera similar, pero el uso de DynamicResource sobre el Lapso objeto aparece en los comentarios:

```
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " DynamicVsStatic.DynamicVsStaticPage "
    Relleno = " 5, 0 ">

    < ContentPage.Resources >
        < ResourceDictionary >
            < x: String x: Key = " CurrentDateTime " > No es en realidad un DateTime </ x: String >
        </ ResourceDictionary >
    </ ContentPage.Resources >

    < StackLayout >
        < Etiqueta Texto = " StaticResource en label.text: "
            VerticalOptions = " EndAndExpand "
            Tamaño de fuente = " Medio " />

        < Etiqueta Texto = " {} StaticResource CurrentDateTime "
            VerticalOptions = " StartAndExpand "
            HorizontalTextAlignment = " Centrar "
            Tamaño de fuente = " Medio " />

        < Etiqueta Texto = " {0} DynamicResource en label.text: "
            VerticalOptions = " EndAndExpand "
            Tamaño de fuente = " Medio " />

        < Etiqueta Texto = " {} DynamicResource CurrentDateTime "
            VerticalOptions = " StartAndExpand "
            HorizontalTextAlignment = " Centrar "
            Tamaño de fuente = " Medio " />

        < Etiqueta Texto = " StaticResource en Span.Text: "
            VerticalOptions = " EndAndExpand "
            Tamaño de fuente = " Medio " />

        < Etiqueta VerticalOptions = " StartAndExpand "
            HorizontalTextAlignment = " Centrar " >
```

```

    Tamaño de fuente = "Medio" >
<Label.FormattedText>
    <FormattedString>
        <Lapso Texto = "{} StaticResource CurrentDateTime" />
    </FormattedString>
</Label.FormattedText>
</Etiqueta >

<!-- Esto plantea una excepción de tiempo de ejecución! -->

<!-- <Etiquetas de texto = "DynamicResource en Span.Text:">
    VerticalOptions = "EndAndExpand"
    Tamaño de Letra = "Medium" />

    <VerticalOptions label = "StartAndExpand"
        HorizontalTextAlignment = "centro"
        Tamaño de Letra = "Medium">
        <Label.FormattedText>
            <FormattedString>
                <Span Text = "{} DynamicResource CurrentDateTime" />
            </FormattedString>
        </Label.FormattedText>
    </Etiquetas> -->
</StackLayout>
</Pagina de contenido >
```

Es probable que esperar los tres de las referencias al `CurrentDateTime` elemento de diccionario para dar lugar a la visualización del texto "No es en realidad un `DateTime`". Sin embargo, el archivo de código subyacente inicia un temporizador en marcha. Cada segundo, la devolución de llamada temporizador sustituye al elemento de diccionario con una nueva cadena que representa una real

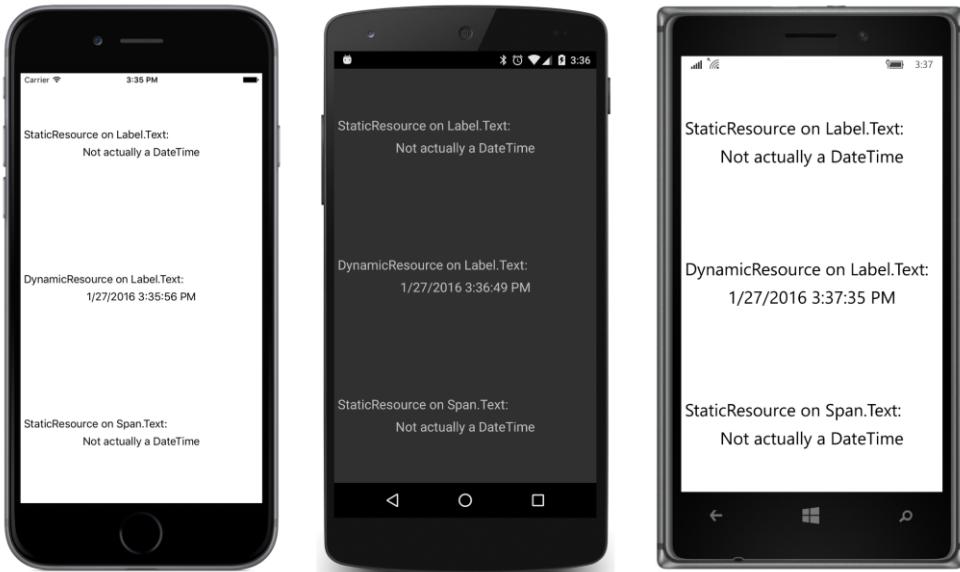
Fecha y hora valor:

```

pública clase parcial DynamicVsStaticPage : Pagina de contenido
{
    pública DynamicVsStaticPage ()
    {
        InitializeComponent ();

        Dispositivo .StartTimer ( Espacio de tiempo .FromSeconds (1),
            () =>
            {
                recursos [ "CurrentDateTime" ] = Fecha y hora .Now.ToString ();
                return true ;
            });
    }
}
```

El resultado es que el Texto propiedades de conjunto con `StaticResource` siendo el mismo, mientras que el que tiene `DynamicResource` cambia cada segundo para reflejar el nuevo elemento en el diccionario:



Aquí hay otra diferencia: si no hay ningún elemento en el diccionario con el nombre clave especificada, Estático-Recurso provocará una excepción en tiempo de ejecución, pero DynamicResource no lo hará.

Usted puede tratar eliminando el comentario del bloque de marcado al final de la **DynamicVsStatic** proyecto, y que de hecho se encontrará con una excepción en tiempo de ejecución en el sentido de que el Texto propiedad no se pudo encontrar. Sólo la ligera, esta excepción no suena del todo bien, pero se está haciendo referencia a una diferencia muy real.

El problema es que la Texto propiedades en Etiqueta y Lapso se define en muy diferentes formas, y esa diferencia tiene mucha importancia para la DynamicResource. Esta diferencia se estudiará en el siguiente capítulo, "La infraestructura enlazable."

extensiones de marcado Minoritarias

Tres extensiones de marcado no se utilizan tanto como los demás. Estos son:

- x: Null
- x: Tipo
- x: Array

Se utiliza el x: Null extensión a establecer una propiedad de nulo. La sintaxis es la siguiente:

```
<SomeElement SomeProperty = "x: Null" />
```

Esto no tiene mucho sentido a menos SomeProperty tiene un valor predeterminado que no es nulo cuando es deseable establecer la propiedad nulo. Sin embargo, como se verá en el capítulo 12, a veces una propiedad puede adquirir una

no-nulo valor de un estilo, y x: Null es prácticamente la única forma de anular eso.

los x: Tipo extensión de marcado se utiliza para establecer una propiedad de tipo Tipo, la clase de .NET que describe el tipo de una clase o estructura. Ésta es la sintaxis:

```
<AnotherElement TypeProperty = " {X: Tipo Color} " />
```

También utilizará x: Tipo en conexión con x: Array. Los x: Array extensión de marcado se utiliza siempre con la sintaxis de elementos regulares en lugar de la sintaxis rizado corsé. Se ha nombrado un argumento necesario Tipo que se establece con el x: Tipo extensión de marcado. Esto indica que el tipo de los elementos de la matriz. Así es como una matriz podría ser definido en un diccionario de recursos:

```
<x: Array x: Key = " formación "
    Tipo = " {X: x Tipo: String} " >
    <x: String > una cuerda <x: String >
    <x: String > dos cuerdas <x: String >
    <x: String > Hilo rojo <x: String >
    <x: String > cadena azul <x: String >
</x: Array >
```

Una extensión de marcado personalizada

Vamos a crear nuestra propia extensión de marcado llamado HslColorExtension. Esto nos permitirá establecer cualquier tipo de propiedad Color especificando valores de matiz, saturación y luminosidad, pero de una manera mucho más simple que el uso de la x: FactoryMethod etiqueta demostró en el capítulo 8, “Código XAML y en armonía.”

Por otra parte, vamos a poner esta clase en una biblioteca de clases portátil independiente para que pueda usarlo desde múltiples aplicaciones. Dicha biblioteca se puede encontrar con el otro código fuente de este libro. Está en un directorio llamado **bibliotecas** que es paralelo a los directorios capítulo aparte. El nombre de este PCL (y el espacio de nombres de las clases dentro de ella) es **Xamarin.FormsBook.Toolkit**.

Puede utilizar esta biblioteca a sí mismo en sus propias aplicaciones mediante la adición de una referencia a él. A continuación, puede añadir una nueva declaración de espacio de nombres XML en sus archivos XAML al igual que para especificar esta biblioteca:

```
xmns: kit de herramientas = "clr-espacio de nombres: Xamarin.FormsBook.Toolkit; montaje = Xamarin.FormsBook.Toolkit"
```

Con este kit de herramientas Prefijo a continuación, puede hacer referencia a la HslColorExtension clase en el mismo modo de usar otras extensiones de marcado XAML:

```
<BoxView Color = " {Toolkit: HslColor H = 0,67, S = 1, L = 0,5} " />
```

A diferencia de otras extensiones de marcado XAML mostrados hasta ahora, éste tiene múltiples propiedades, y si se les está configurando como argumentos con la sintaxis rizado corsé, que debe estar separado por comas.

Sería algo así como que sea útil? Primero vamos a ver cómo crear una biblioteca como para las clases que le gustaría compartir entre aplicaciones:

En Visual Studio, desde el Archivo menú, seleccione Nuevo y Proyecto. En el Nuevo proyecto diálogo, seleccione Visual C# y Cruz-Plataforma a la izquierda, y Biblioteca de clases (Xamarin.Forms) de la lista. Encontrar una ubicación para el proyecto y darle un nombre. Para el PCL creado para este ejemplo, el nombre es

Xamarin.FormsBook.Toolkit. Hacer clic DE ACUERDO. Junto con todos los gastos generales para el proyecto, la plantilla crea un archivo de código Xamarin.FormsBook.Toolkit.cs designado que tiene una clase llamada Xamarin.Forms-Book.Toolkit. Eso no es un nombre de clase válido, por lo que sólo eliminar ese archivo.

En Xamarin de estudio, desde el Archivo menú, seleccione Nuevo y Solución. En el Nuevo proyecto diálogo, seleccione

Multiplataforma y Biblioteca a la izquierda, y formas y Biblioteca de clases de la lista. Encontrar un lugar para él y darle un nombre (Xamarin.FormsBook.Toolkit para este ejemplo). Hacer clic DE ACUERDO. La plantilla de solución crea varios archivos, incluyendo un archivo llamado MyPage.cs. Eliminar ese archivo.

Ahora puede agregar clases para este proyecto de la forma habitual:

En Visual Studio, haga clic en el nombre del proyecto, seleccione Añadir y Nuevo artículo. En el Agregar ítem nuevo de diálogo, si sólo está creando una clase de código solamente, selecciona Visual C# y Código a la izquierda, y seleccione Clase de la lista. Darle un nombre (HslColorExtension.cs para este ejemplo). Haga clic en el Añadir botón.

En Xamarin Studio, en el menú de herramientas para el proyecto, seleccione Añadir y Archivo nuevo. En el Archivo nuevo de diálogo, si sólo está creando una clase de código solamente, selecciona General a la izquierda y Clase vacía en la lista. Darle un nombre (HslColorExtension.cs para este ejemplo). Haga clic en el Nuevo botón.

los Xamarin.FormsBook.Toolkit biblioteca se construyó y se acumulan clases útiles durante el curso de este libro. Pero la primera clase en esta biblioteca es HslColorExtension. El archivo HslColorExtension.cs (incluyendo la requerida utilizando directivas) se ve así:

```
utilizando Sistema;
utilizando Xamarin.Forms;
utilizando Xamarin.Forms.Xaml;

espacio de nombres Xamarin.FormsBook.Toolkit
{
    clase pública HslColorExtension : IMarkupExtension
    {
        público HslColorExtension ()
        {
            A = 1;
        }

        pública doble H { conjunto ; obtener ; }

        pública doble S { conjunto ; obtener ; }

        pública doble L { conjunto ; obtener ; }

        pública doble UN { conjunto ; obtener ; }

        objeto público ProvideValue ( IServiceProvider proveedor de servicio)
        {
            regreso Color .FromHsla (H, S, L, A);
        }
    }
}
```

```
    }
}
}
```

Observe que la clase es pública, por lo que es visible desde fuera de la biblioteca, y que implementa el **IMarkupExtension** de la interfaz, lo que significa que debe incluir una **ProvideValue** método. Sin embargo, el método no hace uso de la **IServiceProvider** argumento en absoluto, sobre todo porque no necesita saber acerca de cualquier otra cosa externa a sí mismo. Todas las necesidades de TI son las cuatro propiedades para crear una **Color** valor, y si el UN valor no se establece, se utiliza un valor predeterminado de 1 (totalmente opaco).

Esta **Xamarin.FormsBook.Toolkit** solución contiene sólo un proyecto PCL. El proyecto puede ser construido para generar un conjunto de PCL, pero no se puede ejecutar sin una aplicación que utilice este montaje.

Hay dos formas de acceder a esta biblioteca a partir de una solución de aplicación:

- Desde el proyecto PCL de su solución de aplicación, agregue una referencia al ensamblado de biblioteca PCL, que es la biblioteca de vínculos dinámicos (DLL) generado a partir del proyecto de la biblioteca.
- Para incluir un enlace al proyecto de biblioteca de su solución de aplicación, y añadir una referencia a ese proyecto de la biblioteca del proyecto PCL de la applicationt.

La primera opción es necesaria si usted tiene sólo el DLL y no el proyecto con el código fuente. Tal vez usted está obteniendo licencia para la biblioteca y no tienen acceso a la fuente. Pero si usted tiene acceso al proyecto, por lo general es mejor para incluir un enlace al proyecto de biblioteca en su solución para que pueda realizar cambios fácilmente en el código de la biblioteca y reconstruir el proyecto de la biblioteca.

El proyecto final de este capítulo es **CustomExtensionDemo**, que hace uso de la **HslColorExtension** clase en la nueva biblioteca. los **CustomExtensionDemo** solución contiene un enlace a la **Xamarin.FormsBook.Toolkit** proyecto PCL y la referencias sección en el **CustomExtensionDemo** enumera el proyecto **Xamarin.FormsBook.Toolkit** montaje.

Ahora el proyecto de aplicación que parecía estar dispuesto a acceder al proyecto de biblioteca para usar la **HslColorExtension** clase dentro de archivo XAML de la aplicación.

Pero primero hay otro paso. A menos que haya habilitado la compilación XAML, una referencia a una biblioteca externa de XAML es insuficiente para asegurar que la biblioteca se incluye con la aplicación. La biblioteca necesita ser visitada a partir del código real. Por esta razón, **Xamarin.FormsBook.Toolkit** también contiene una clase y un método que podría parecer a partir del nombre de ser importante realizar la inicialización de la biblioteca:

```
espacio de nombres Xamarin.FormsBook.Toolkit
{
    público clase estática Toolkit
    {
        public static void En eso()
        {
        }
    }
}
```

Siempre que utilice cualquier cosa de esta biblioteca, tratar de adquirir la costumbre de llamar a este En eso Método primera hora de la Aplicación archivo:

```
espacio de nombres CustomExtensionDemo
{
    clase pública Aplicación : Solicitud
    {
        público App ()
        {
            Xamarin.FormsBook.Toolkit.Toolkit .En eso();

            MainPage = nuevo CustomExtensionDemoPage ();
        }
        ...
    }
}
```

El archivo XAML siguiente se muestra la declaración de espacio de nombres XML para el `Xamarin.FormsBook.Toolkit` biblioteca y tres formas de acceder al marcado XAML encargo de extensión mediante el uso de una `HslColorExtension` Conjunto de elementos de la sintaxis de la propiedad de elementos en el `Color` propiedad y mediante el uso de ambos `HslColorExtension` y `HslColor` con la sintaxis rizado corsé más común. Una vez más, observe el uso de comas para separar los argumentos dentro de las llaves:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    xmlns: kit de herramientas =
        "clr-espacio de nombres: Xamarin.FormsBook.Toolkit; montaje = Xamarin.FormsBook.Toolkit "
    x: Class = " CustomExtensionDemo.CustomExtensionDemoPage " >

    < StackLayout >
        <! - rojo ->
        < BoxView HorizontalOptions = " Centrar "
            VerticalOptions = " CenterAndExpand " >
            < BoxView.Color >
                < kit de herramientas: HslColorExtension H = " 0 " S = " 1 " L = " 0.5 " />
            </ BoxView.Color >
        </ BoxView >

        <! - Verde ->
        < BoxView HorizontalOptions = " Centrar "
            VerticalOptions = " CenterAndExpand " >
            < BoxView.Color >
                < kit de herramientas: HslColorExtension H = " 0.33 " S = " 1 " L = " 0.5 " />
            </ BoxView.Color >
        </ BoxView >

        <! - Azul ->
        < BoxView Color = " [Toolkit: HslColor H = 0.67, S = 1, L = 0.5] "
            HorizontalOptions = " Centrar "
            VerticalOptions = " CenterAndExpand " />

        <! - gris ->
        < BoxView Color = " [Toolkit: HslColor H = 0, S = 0, L = 0.5] " />
```

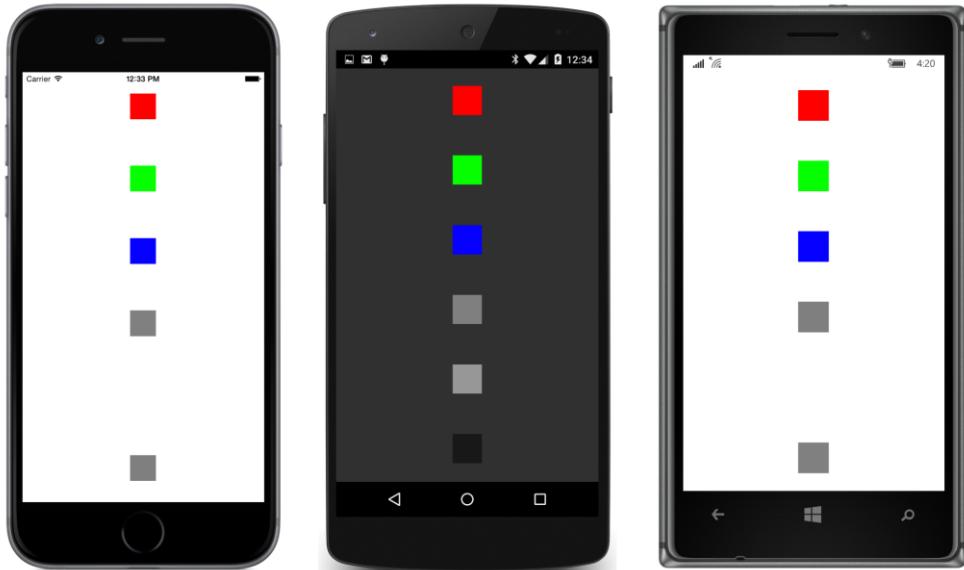
```
    HorizontalOptions = "Centrar"
    VerticalOptions = "CenterAndExpand" />

    <!-- blanca semitransparente -->
    <BoxView Color = "{Toolkit: HslColor H = 0, S = 0, L = 1, A = 0,5}"*
        HorizontalOptions = "Centrar"
        VerticalOptions = "CenterAndExpand" />

    <!-- negro semitransparente -->
    <BoxView Color = "{Toolkit: HslColor H = 0, S = 0, L = 0, A = 0,5}"*
        HorizontalOptions = "Centrar"
        VerticalOptions = "CenterAndExpand" />

</StackLayout>
</Pagina de contenido>
```

Los dos últimos ejemplos establecen el UN viviendo en un 50 por ciento la transparencia, por lo que las cajas se muestran como un tono de gris (o en absoluto) en función de los antecedentes:



Dos usos principales de extensiones de marcado XAML están aún por venir. En el capítulo 12, verá el Estilo clase, que es sin duda el tema más popular para la inclusión en los diccionarios de recursos, y en el capítulo 16, verá la poderosa extensión de marcado llamado Unión.

Capítulo 11

La infraestructura enlazable

Una de las construcciones básicas del lenguaje de C # es el miembro de la clase conocida como la *propiedad*. Todos nosotros muy pronto en nuestros primeros encuentros con C # aprendido la rutina general de la definición de una propiedad. La propiedad es a menudo respaldado por un campo privado e incluye conjunto y obtener descriptores de acceso que hacen referencia al ámbito privado y hacer algo con un nuevo valor:

```
clase pública Mi clase
{
    ...
    doble calidad;

    pública doble Calidad
    {
        conjunto
        {
            calidad = valor;
            // hacer algo con el nuevo valor
        }
        obtener
        {
            regreso calidad;
        }
    ...
}
```

Las propiedades se denominan a veces *campos inteligentes*. Sintácticamente, el código que tiene acceso a una propiedad similar a código que tiene acceso a un campo. Sin embargo, la propiedad puede ejecutar algunos de su propio código cuando se accede a la propiedad.

Las propiedades son también métodos similares. De hecho, el código C # es compilado en lenguaje intermedio que implementa una propiedad como Calidad con un par de métodos llamado set_Quality y get_Quality. Sin embargo, a pesar de la semejanza funcional cerca entre propiedades y un par de conjunto y obtener métodos, la sintaxis de la propiedad se revela a ser mucho más adecuado cuando se pasa de código para el mercado. Es difícil imaginar XAML construido sobre una API subyacente que falta propiedades.

Así que es posible que se sorprenda al saber que Xamarin.Forms implementa una definición de propiedad mejorada que se basa en las propiedades de C#. O tal vez usted no se sorprenderá. Si ya tiene experiencia con las plataformas basadas en XAML de Microsoft, podrás encontrar algunos conceptos familiares en este capítulo.

La definición de la propiedad se muestra más arriba se conoce como una *propiedad CLR* porque está apoyada por el tiempo de ejecución de lenguaje común .NET. La definición de la propiedad mejorada en Xamarin.Forms se basa en la propiedad de CLR y se llama una *propiedad que puede vincularse*, encapsulado por la BindableProperty clase y el apoyo de la bindableObject clase.

La jerarquía de clases Xamarin.Forms

Antes de explorar los detalles de la importante `bindableObject` clase, vamos a descubrir como el primero `BindableObject` encaja en la arquitectura Xamarin.Forms en general mediante la construcción de una jerarquía de clases.

En un marco de programación orientado a objetos tal como Xamarin.Forms, una jerarquía de clases a menudo puede revelar importantes estructuras internas del medio ambiente. La jerarquía de clases muestra cómo las diversas clases se relacionan entre sí y las propiedades, métodos y eventos que comparten, incluyendo la forma en que pueden vincularse propiedades son compatibles.

Se puede construir una jerarquía de dicha clase por laboriosamente pasando por la documentación en línea y tomando nota de lo que las clases se derivan de lo que otras clases. O puede escribir un programa Xamarin.Forms para hacer el trabajo para usted y para mostrar la jerarquía de clases en el teléfono. Dicho programa hace uso de .NET reflexión para obtener todos los públicos clases, estructuras y enumeraciones de la **Xamarin.Forms.Core** y **Xamarin.Forms.Xaml** asambleas y disponerlas en un árbol. Los **ClassHierarchy**

la aplicación muestra esta técnica.

Como de costumbre, el **ClassHierarchy** proyecto contiene una clase que deriva de `Página de contenido`, llamado `ClassHierarchyPage`, pero también contiene dos clases adicionales, llamados `TypeInformation` y `ClassAndSubclasses`.

El programa crea una `TypeInformation` instancia para cada clase pública (y la estructura y la enumeración) en el **Xamarin.Forms.Core** y **Xamarin.Forms.Xaml** montajes, además de cualquier clase de .NET que sirve como una clase base para cualquier clase **Xamarin.Forms**, con la excepción de `Objeto`. (Estas clases son .NET `Atributo`, `Delegado`, `Enum`, `EventArgs`, `Excepción`, `MulticastDelegate`, y `Tipo` de valor.)

Los `TypeInformation` constructor requiere una `Tipo` objeto que identifica un tipo sino que también obtiene alguna otra información:

```
clase TypeInformation
{
    bool isBaseGenericType;
    Tipo baseGenericTypeDef;

    público TypeInformation ( Tipo tipo, bool isXamarinForms )
    {
        Type = tipo;
        IsXamarinForms = isXamarinForms;
        TypeInfo TypeInfo = type.GetTypeInfo ();
        BaseType = TypeInfo.BaseType;

        Si (BaseType = nulo )
        {
            TypeInfo baseTypeInfo = BaseType.GetTypeInfo ();
            isBaseGenericType = baseTypeInfo.IsGenericType;

            Si (IsBaseGenericType)
            {
                baseGenericTypeDef = baseTypeInfo.GetGenericTypeDefinition ();
            }
        }
    }
}
```

```

    }
}

}

pùblico Tipo Tipo { conjunto privado ; obtener ; }

pùblico Tipo BaseType { conjunto privado ; obtener ; }

public bool IsXamarinForms { conjunto privado ; obtener ; }

public bool IsDerivedDirectlyFrom ( Tipo ParentType)

{

    Si (BaseType1 = nulo && isBaseGenericType)

    {

        Si (BaseGenericTypeDef == ParentType)

        {

            return true ;

        }

    }

    else if (BaseType == ParentType)

    {

        return true ;

    }

    falso retorno ;

}

}

```

Una parte muy importante de esta clase es el `IsDerivedDirectlyFrom` método, que devolverá cierto

si se pasa un argumento que es el tipo de base de este tipo. Esta determinación se complica si se trata de clases genéricas, y que tema explica en gran medida la complejidad de la clase.

los ClassAndSubclasses clase es considerablemente más corto:

```
clase ClassAndSubclasses

{
    público (ClassAndSubclasses Tipo padre, bool IsXamarinForms)
    {
        Type = padre;
        IsXamarinForms = IsXamarinForms;
        subclases = nuevo Lista < ClassAndSubclasses >();
    }

    público Tipo Tipo { conjunto privado ; obtener ; }
    público bool IsXamarinForms { conjunto privado ; obtener ; }
    público Lista < ClassAndSubclasses > subclases { conjunto privado ; obtener ; }
}
```

El programa crea una instancia de esta clase para cada Tipo que se muestra en la jerarquía de clases, incluyendo Objeto, por lo que el programa crea una más ClassAndSubclasses ejemplo que el número de TypeInformation instancias. Los ClassAndSubclasses instancia asociada con Objeto contiene una colección de todas las clases que se derivan directamente de Objeto, y cada uno de aquellos ClassAndSubclasses casos contiene una colección de todas las clases que se derivan de que uno, y así sucesivamente para el resto del árbol de jerarquía.

los ClassHierarchyPage clase consiste en un archivo XAML y un archivo de código subyacente, pero el archivo XAML contiene poco más que un desplazable StackLayout listo para algunos Etiqueta elementos:

```
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns:x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " ClassHierarchy.ClassHierarchyPage ">

    < ContentPage.Padding >
        < OnPlatform x: TypeArguments = " Espesor "
            iOS = " 5, 20, 0, 0 "
            Androide = " 5, 0, 0, 0 "
            WinPhone = " 5, 0, 0, 0 " />
    </ ContentPage.Padding >

    < ScrollView >
        < StackLayout x: Nombre = " stackLayout "
            Espaciado = " 0 " />
    </ ScrollView >
</ Pagina de contenido >
```

El archivo de código subyacente obtiene referencias a los dos Xamarin.Forms Montaje objetos y luego se acumula todas las clases públicas, estructuras y enumeraciones de la Lista de clase colección. A continuación, comprueba por la necesidad de incluir ningún clases base de los conjuntos de .NET, ordena el resultado, y entonces llama a dos métodos recursivos, AddChildrenToParent y AddItemToStackLayout:

```
público clase parcial ClassHierarchyPage : Pagina de contenido
{
    público ClassHierarchyPage ()
    {
        InitializeComponent ();

        Lista < TypeInformation > = classList nuevo Lista < TypeInformation > 0;

        // Obtener tipos de ensamblaje Xamarin.Forms.Core.
        (GetPublicTypes tipo de ( Ver ) .GetTypeInfo () de la Asamblea, classList);

        // Obtener tipos de ensamblaje Xamarin.Forms.Xaml.
        (GetPublicTypes tipo de ( extensiones ) .GetTypeInfo () de la Asamblea, classList);

        // Asegúrese de que todas las clases tienen un tipo de base en la lista.
        // (es decir, añadir Attribute, ValueType, Enum, EventArgs, etc.)
        En t index = 0;

        // ¡Cuidado! Bucle mediante la expansión de classList!
        hacer
        {
            // Obtener un tipo infantil de la lista.
            TypeInformation childType = classList [indice];

            Si (ChildType.TypeId = tipo de ( Objeto ))
            {
                bool hasBaseType = falso ;
            }
        }
    }
}
```

```

// Recorrer la lista en busca de un tipo base.
para cada ( TypeInformation ParentType en Lista de clase)
{
    Si (ChildType.IsDerivedDirectlyFrom (parentType.Type))
    {
        has BaseType = cierto ;
    }
}

// Si no hay ningún tipo de base, añadirlo.
Si (! HasBaseType && childType.BaseType = tipo de ( Objeto ))
{
    classList.Add ( nuevo TypeInformation (childType.BaseType, falso ) );
}
}

indice++;
}

mientras (indice <classList.Count);

// Ahora ordenar la lista.
classList.Sort ((t1, t2) =>
{
    regreso Cuerda ¡Compare (t1.Type.Name, t2.Type.Name);
});

// Inicia la pantalla con System.Object.
ClassAndSubclasses rootClass = nuevo ClassAndSubclasses ( tipo de ( Objeto ), falso );

// método recursivo para construir el árbol de jerarquía.
AddChildrenToParent (rootClass, classList);

// método recursivo para añadir elementos a StackLayout.
AddItemToStackLayout (rootClass, 0);
}

vacío (GetPublicTypes Montaje montaje,
       Lista < TypeInformation > ClassList)
{
    // bucle a través de todos los tipos.
    para cada ( Tipo tipo en assembly.ExportedTypes )
    {
        TypeInfo TypeInfo = type.GetTypeInfo ();

        // tipos públicos sólo, pero excluyen las interfaces.
        Si (TypeInfo.IsPublic && ! TypeInfo.IsInterface)
        {
            // Añadir a la lista Tipo.
            classList.Add ( nuevo TypeInformation (tipo, cierto ) );
        }
    }
}

vacío AddChildrenToParent ( ClassAndSubclasses ParentClass,
                           Lista < TypeInformation > ClassList)

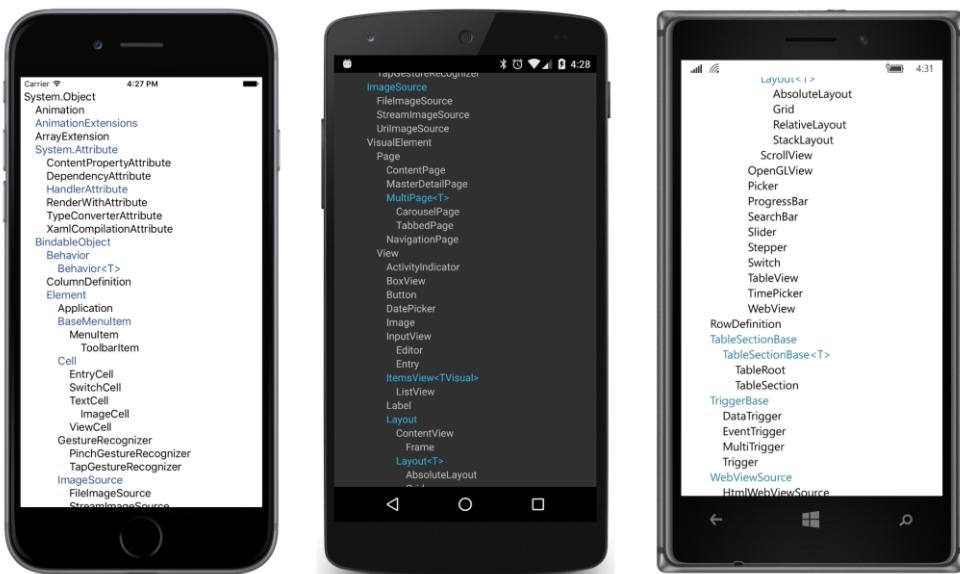
```

```
{  
    para cada ( TypeInformation typeInformation en Lista de clase)  
    {  
        Si (TypeInformation.IsDerivedDirectlyFrom (parentClass.Type))  
        {  
            ClassAndSubclasses subclass =  
                nuevo ClassAndSubclasses (TypeInformation.Type,  
                    typeInformation.IsXamarinForms);  
            parentClass.Subclasses.Add (subclase);  
            AddChildrenToParent (subclase, classList);  
        }  
    }  
}  
  
vacío AddItemToStackLayout ( ClassAndSubclasses ParentClass, Ent nivel)  
{  
    // Si el conjunto no es Xamarin.Forms, mostrar el nombre completo.  
    cuerda name = parentClass.IsXamarinForms? parentClass.Type.Name:  
        parentClass.Type.FullName;  
  
    TypeInfo TypeInfo = parentClass.Type.GetTypeInfo ();  
  
    // Si, soportes y parámetros genéricos ángulo de la pantalla.  
    Si (TypeInfo.IsGenericType)  
    {  
        Tipo [] Parámetros = typeInfo.GenericTypeParameters;  
        name = name.Substring (0, name.Length - 2);  
        nombrar += "<";  
  
        para ( Ent i = 0; i <parameters.Length; i ++)  
        {  
            nombre += parámetros [i].Name;  
            Si (i <parameters.Length - 1)  
            {  
                nombrar += " ";  
            }  
        }  
        nombrar += ">";  
    }  
  
    // Crear etiqueta y añadir a StackLayout.  
    Etiqueta etiqueta = nuevo Etiqueta  
    {  
        text = Cuerda .Formato ("{0} {1}", nueva cadena (* , 4 *), nombre),  
        TextColor = parentClass.Type.GetTypeInfo ().IsAbstract?  
            Color .Accent: Color .Defecto  
        };  
  
        stackLayout.Children.Add (etiqueta);  
  
        // Ahora mostrar los tipos anidados.  
        para cada ( ClassAndSubclasses subclase en parentClass.Subclasses)  
        {  
            AddItemToStackLayout (subclase, nivel + 1);  
        }  
}
```

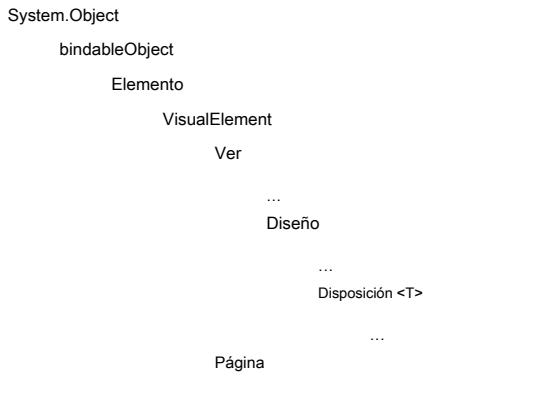
}

el recursiva AddChildrenToParent método ensambla la lista enlazada de ClassAndSubclasses

instancias de la plena Lista de clase colección. Los AddItemToStackLayout método también es recursiva, ya que es responsable de añadir la ClassesAndSubclasses lista vinculada a la StackLayout objeto mediante la creación de una Etiqueta ver para cada clase, con un poco de espacio en blanco al principio para el sangrado adecuado. El método muestra los tipos Xamarin.Forms con sólo los nombres de las clases, pero los tipos de .NET incluye el nombre completo para distinguirlos. El método utiliza el color de acento plataforma para las clases que no son replicables porque son abstractos o estática:



En general, verá que los Xamarin.Forms elementos visuales tienen la siguiente jerarquía en general:



A parte de Objeto, todas las clases en esta jerarquía de clases abreviado se implementan en el conjunto Xamarin.Forms.Core.dll y asociados con un espacio de nombres de Xamarin.Forms.

Vamos a examinar algunas de estas clases principales en detalle.

Como el nombre de la bindableObject clase implica, la función primaria de esta clase es para apoyar la unión de dos propiedades de dos objetos del enlace de datos de manera que mantengan el mismo valor. Pero bindableObject también es compatible con los estilos y la DynamicResource extensión de marcado también. Esto se hace de dos maneras: a través de bindableObject definiciones de propiedades en la forma de BindableProperty objetos y también por la aplicación de la NET INotifyPropertyChanged interfaz. Todo esto se discutirá con más detalle en este capítulo y en los capítulos futuros.

Vamos a continuar en la jerarquía: como hemos visto, los objetos de interfaz de usuario en Xamarin.Forms a menudo se organizan en la página en una jerarquía de elementos primarios y secundarios, y el Elemento clase incluye soporte para las relaciones de padres e hijos.

VisualElement es una clase de excepcional importancia en Xamarin.Forms. Un elemento visual es nada en Xamarin.Forms que ocupa un área en la pantalla. Los VisualElement clase define 28 propiedades públicas relacionadas con el tamaño, la ubicación, color de fondo, y otras características visuales y funcionales, tales como Está habilitado y Es visible.

En Xamarin.Forms la palabra *Ver* a menudo se utiliza para referirse a objetos visuales individuales, tales como botones, deslizadores y cajas de entrada de texto, pero se puede ver que el Ver clase es la de los padres a las clases de diseño también. Curiosamente, Ver agrega sólo tres miembros públicos a lo que se hereda de VisualElement. Estos son HorizontalOptions y VerticalOptions -que tiene sentido debido a que estas propiedades no se aplican a las páginas y GestureRecognizers para apoyar la entrada táctil.

Los descendientes de Diseño son capaces de tener puntos de vista de los niños. Una vista del niño aparece en la pantalla visualmente dentro de los límites de su padre. Las clases que se derivan de Diseño puede tener un solo hijo de tipo Ver, pero el genérico Disposición <T> clase define una Niños propiedad, que es un conjunto de múltiples puntos de vista del niño, incluyendo otros diseños. Ya ha visto la StackLayout, que organiza sus hijos en una pila horizontal o vertical. Aunque el Diseño clase se deriva de Ver, los diseños son tan importantes en Xamarin.Forms que a menudo se consideran una categoría en sí mismos.

ClassHierarchy enumera todas las clases públicas, estructuras y enumeraciones definidas en el Xamarin.Forms.Core y Xamarin.Forms.Xaml asambleas, pero no enumera las interfaces. Los que son importantes también, pero vas a tener que explorar por su cuenta. (O mejorar el programa para enumerarlos.)

Ni tampoco ClassHierarchy enumerar las muchas clases públicas que ayudan a poner en práctica Xamarin.Forms en las distintas plataformas. En el capítulo final de este libro, verá una versión que hace.

Una ojeada en bindableObject y BindableProperty

La existencia de clases llamado `bindableObject` y `BindableProperty` es probable que sea un poco complicado al principio. Manten eso en mente `bindableObject` es muy parecido Objeto en que sirve como una clase base para una gran parte de la API Xamarin.Forms, y en particular a Elemento y por lo tanto `VisualElement`.

`bindableObject` proporciona soporte para objetos de tipo `BindableProperty`. UN `BindableProperty`

objeto se extiende una propiedad CLR. Las mejores perspectivas sobre propiedades enlazables vienen cuando se crea un poco de lo propio, como se va a realizar antes del final de este capítulo, pero también se puede deducir una cierta comprensión mediante la exploración de las propiedades enlazables existentes.

Hacia el comienzo del capítulo 7, "código XAML frente," dos botones se crearon con muchos de los mismos valores de propiedades, excepto que las propiedades de un botón se establecieron en el código utilizando la sintaxis de inicialización C # 3.0 objeto y el otro botón se crea una instancia e inicializado en XAML.

He aquí una similar (pero sólo de código) programa llamado `PropertySettings` que también crea e inicializa dos botones de dos maneras diferentes. Las propiedades de la primera Etiqueta se establecen la manera antigua, mientras que las propiedades del segundo Etiqueta se fijó con una técnica más detallado:

```
clase pública PropertySettingsPage : Pagina de contenido
{
    público PropertySettingsPage ()
    {
        Etiqueta label1 = nuevo Etiqueta ();
        Label1.Text = "Texto propiedades CLR";
        label1.isVisible = cierto ;
        label1.Opacity = 0,75;
        label1.HorizontalTextAlignment = Alineación del texto .Centrar;
        label1.VerticalOptions = LayoutOptions .CenterAndExpand;
        label1.TextColor = Color .Azul;
        label1.BackgroundColor = Color .FromRgb (255, 128, 128);
        label1.FontSize = Dispositivo .GetNamedSize ( NamedSize .Medio, nuevo Etiqueta ());
        label1.FontAttributes = FontAttributes .bold | FontAttributes .Itálico;

        Etiqueta label2 = nuevo Etiqueta ();
        label2.SetValue ( Etiqueta .TextProperty, "Texto con propiedades enlazables");
        label2.SetValue ( Etiqueta .isVisibleProperty, cierto );
        label2.SetValue ( Etiqueta .OpacityProperty, 0,75);
        label2.SetValue ( Etiqueta .HorizontalTextAlignmentProperty, Alineación del texto .Centrar);
        label2.SetValue ( Etiqueta .VerticalOptionsProperty, LayoutOptions .CenterAndExpand);
        label2.SetValue ( Etiqueta .TextColorProperty, Color .Azul);
        label2.SetValue ( Etiqueta .BackgroundColorProperty, Color .FromRgb (255, 128, 128));
        label2.SetValue ( Etiqueta .FontSizeProperty,
            Dispositivo .GetNamedSize ( NamedSize .Medio, nuevo Etiqueta ()));
        label2.SetValue ( Etiqueta .FontAttributesProperty,
            FontAttributes .bold | FontAttributes .Itálico);

        content = nuevo StackLayout
    }
}
```

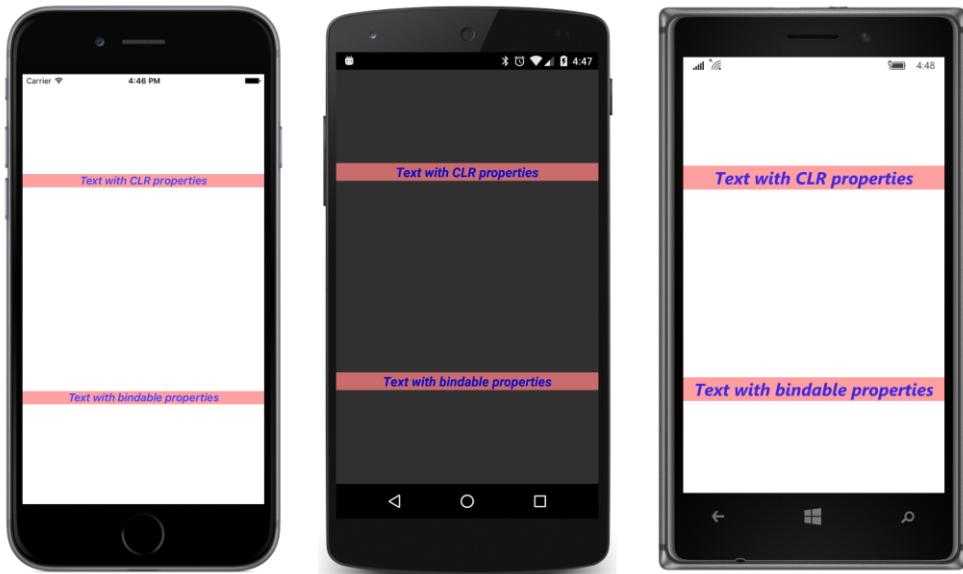
```

    Los niños =
    {
        label1,
        label2
    }
};

}
}

```

Estas dos formas de configurar las propiedades son totalmente coherentes:



Sin embargo, la sintaxis alternativa parece muy extraño. Por ejemplo:

```
label2.SetValue ( Etiqueta .TextProperty, "Texto con propiedades enlazables" );
```

Que es eso Valor ajustado ¿método? Valor ajustado es definido por bindableObject, de la que cada objeto visual deriva. bindableObject también define una GetValue método.

Ese primer argumento Valor ajustado tiene el nombre Label.TextProperty, lo que indica que

TextProperty es estático, pero a pesar de su nombre, no es una propiedad en absoluto. Es una estática campo del Etiqueta clase. TextProperty También es de sólo lectura, y se define en el Etiqueta clase algo como esto:

```
publco estatico solo lectura BindableProperty TextProperty;
```

Eso es un objeto de tipo BindableProperty. Por supuesto, puede parecer un poco preocupante que un campo se denomina TextProperty, pero no lo es.

Porque es estático, sin embargo, que existe independientemente de cualquier Etiqueta objetos que podría o no podría existir.

Si nos fijamos en la documentación de la Etiqueta clase, verá que define 10 propiedades, incluyendo Texto, Color de texto, Tamaño de fuente, FontAttributes, y otros. También verá 10 correspondientes

campos de sólo lectura public static de tipo BindableProperty con los nombres TextProperty, TextColorProperty, FontSizeProperty, FontAttributesProperty, Etcétera.

Estas propiedades y campos están estrechamente relacionados. De hecho, interna a la Etiqueta clase, el Texto propiedad CLR se define así para hacer referencia al correspondiente TextProperty objeto:

```
public cuerda Texto
{
    conjunto { Valor ajustado( Etiqueta .TextProperty, valor ); }
    obtener { regreso ( cuerda ) GetValue ( Etiqueta .TextProperty); }
}
```

Así que ya ves por qué es que su aplicación de llamada Valor ajustado con un Label.TextProperty argumento es exactamente equivalente a establecer la Texto inmueble directamente, y tal vez sólo un poco más rápido más pequeño!

La definición interna de la Texto bienes en Etiqueta No es información secreta. Este es el código estándar. Aunque cualquier clase puede definir una BindableProperty objeto, solamente una clase que deriva de BindableObject puede llamar al Valor ajustado y GetValue métodos que realmente implementan la propiedad de la clase. Se requiere casting para la GetValue método porque se define como regresar objeto.

Todo el trabajo real implicado con el mantenimiento de la Texto propiedad está pasando en los Valor ajustado y GetValue llamadas. Los bindableObject y BindableProperty objetos efectivamente extender la funcionalidad de propiedades CLR estándar para proporcionar una forma sistemática a:

- Definición de las propiedades
- Dar propiedades valores por defecto
- Almacenar sus valores actuales
- Proporcionar mecanismos para la validación de los valores de propiedad
- Mantener la coherencia entre las propiedades relacionadas en una sola clase
- Responder a los cambios de propiedad
- Desencadenar notificaciones cuando una propiedad está a punto de cambiar y ha cambiado
- el enlace de datos de apoyo
- estilos de apoyo
- Apoyar los recursos dinámicos

La estrecha relación de una propiedad denominada Texto con un BindableProperty llamado TextProperty se refleja en la forma en que los programadores hablan de estas propiedades: A veces un programador dice que la Texto propiedad está "respaldada por" una BindableProperty llamado TextProperty porque

TextProperty proporciona apoyo a la infraestructura de Texto. Pero un acceso directo común es decir que Texto es en sí misma una "propiedad enlazable", y por lo general nadie se confundirá.

No todas las propiedades Xamarin.Forms es una propiedad enlazable. Ni el Contenido propiedad de Pagina de contenido ni el Niños propiedad de Disposición <T> es una propiedad enlazable. De las 28 propiedades definidas por VisualElement, 26 están respaldados por propiedades enlazables, pero el Límites propiedad y el recursos propiedades no son.

los Lapso clase usado en conexión con FormattedString no se deriva de BindableObject. Por lo tanto, Lapso no hereda Valor ajustado y GetValue métodos, y no se puede poner en práctica BindableProperty objetos.

Esto significa que el Texto propiedad de Etiqueta está respaldado por una propiedad que puede vincularse, pero el Texto propiedad de Lapso no es. ¿Hace alguna diferencia?

Por supuesto que hace la diferencia! Si recuerdan la **DynamicVsStatic** programa en el capítulo anterior, se descubrió que DynamicResource trabajado en el Texto propiedad de Etiqueta pero no el Texto propiedad de Lapso. ¿Puede ser que DynamicResource sólo funciona con propiedades enlazables?

Esta suposición es más o menos confirmada por la definición de la siguiente método público definido por Elemento:

```
public void SetDynamicResource ( BindableProperty propiedad, cuerda llave);
```

Así es como una clave de diccionario está asociado con una propiedad particular de un elemento cuando esa propiedad es el blanco de una DynamicResource extensión de marcado.

Esta SetDynamicResource método también permite establecer un vínculo dinámico de recursos en una propiedad en el código. Aquí está la clase de página de una versión de sólo código de **DynamicVsStatic** llamado **DynamicVsStaticCode**. Es algo simplificada para excluir el uso de una FormattedString y Lapso objeto, pero por lo demás que imita bastante exactitud cómo se analiza el archivo XAML anterior y, en particular, cómo el Texto propiedades de la Etiqueta elementos son fijados por el analizador XAML:

```
pública clase DynamicVsStaticCodePage : Pagina de contenido
{
    pública DynamicVsStaticCodePage ()
    {
        Relleno = nuevo Espesor (5, 0);

        // Crear diccionario de recursos y añadir el artículo.
        recursos = nuevo ResourceDictionary
        {
            {"CurrentDateTime", "No es en realidad un DateTime"}
        };

        content = nuevo StackLayout
        {
            Los niños =
            {
                nuevo Etiqueta
                {
                    text = "StaticResource en label.text:" ,
                    VerticalOptions = LayoutOptions .EndAndExpand,
                }
            }
        };
    }
}
```

```

        Tamaño de Letra = Dispositivo .GetNamedSize ( NamedSize .Medio, tipo de ( Etiqueta ))
    },

    nuevo Etiqueta
    {
        Text = ( cuerda [Recursos] "CurrentDateTime" ),
        VerticalOptions = LayoutOptions .StartAndExpand,
        HorizontalTextAlignment = Alineación del texto .Centrar,
        Tamaño de Letra = Dispositivo .GetNamedSize ( NamedSize .Medio, tipo de ( Etiqueta ))
    },

    nuevo Etiqueta
    {
        text = "DynamicResource en label.text:" ,
        VerticalOptions = LayoutOptions .EndAndExpand,
        Tamaño de Letra = Dispositivo .GetNamedSize ( NamedSize .Medio, tipo de ( Etiqueta ))
    }
}

};

// crear la etiqueta final con la dinámica de recursos.
Etiqueta etiqueta = nuevo Etiqueta
{
    VerticalOptions = LayoutOptions .StartAndExpand,
    HorizontalTextAlignment = Alineación del texto .Centrar,
    Tamaño de Letra = Dispositivo .GetNamedSize ( NamedSize .Medio, tipo de ( Etiqueta ))
};

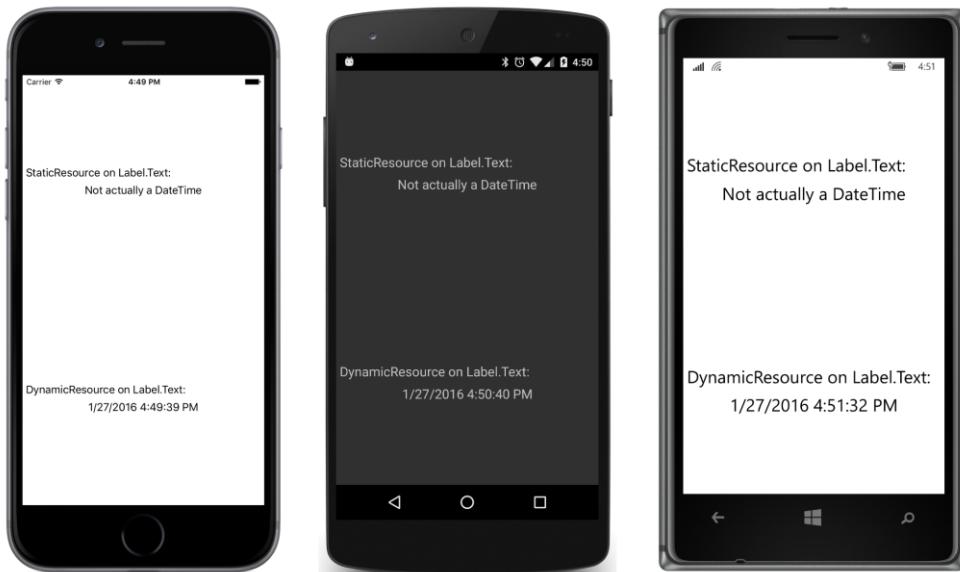
label.SetDynamicResource ( Etiqueta .TextProperty, "CurrentDateTime" );

(( StackLayout ) Contenido) .Children.Add (etiqueta);

// Inicia el temporizador en marcha.
Dispositivo .StartTimer ( Espacio de tiempo .FromSeconds (1),
    ()=>
    {
        recursos [ "CurrentDateTime" ]= Fecha y hora .Now.ToString ();
        return true ;
    });
}
}

```

los Texto propiedad de la segunda Etiqueta se establece directamente desde la entrada del diccionario y hace que el uso del diccionario parece un poco sin sentido en este contexto. Pero el Texto propiedad de la última Etiqueta está unido a la clave de diccionario a través de una llamada a SetDynamicResource, que permite a la propiedad que se actualiza cuando el contenido del diccionario cambian:



Considere esto: ¿Cómo sería la firma de este `SetDynamicResource` método sea si no podía referirse a un establecimiento en el `BindableProperty` *¿objeto?* Es fácil hacer referencia a una propiedad *valor* en llamadas a métodos, pero no la propiedad en sí. Hay un par de maneras, tales como la `PropertyInfo` clase en el `System.Reflection` espacio de nombres o el LINQ Expresión objeto. Pero el `BindableProperty`

objeto está diseñado específicamente para este fin, así como el trabajo esencial de manejo de la relación subyacente entre la propiedad y la clave de diccionario.

Del mismo modo, cuando exploramos estilos en el siguiente capítulo, se encontrará con una `Setter` la clase utilizada en relación con los estilos. `Setter` define una propiedad denominada `Propiedad de tipo BindableProperty`, que establece que cualquier propiedad dirigida por un estilo debe estar respaldada por una propiedad enlazable. Esto permite que un estilo que se define antes de los elementos seleccionados por el estilo.

Lo mismo sucede con los enlaces de datos. Los `bindableObject` clase define una `SetBinding` método que es muy similar a la `SetDynamicResource` método definido en Elemento:

```
public void SetBinding ( BindableProperty targetProperty, BindingBase Unión);
```

Una vez más, observe el tipo del primer argumento. Cualquier propiedad dirigida por un enlace de datos debe estar respaldada por una propiedad enlazable.

Por estas razones, cada vez que se crea una vista personalizada y la necesidad de definir las propiedades públicas, su inclinación por defecto debería ser definir como propiedades enlazables. Sólo si después de una cuidadosa consideración llega a la conclusión de que no es necesario o apropiado para la propiedad a ser el blanco de un estilo o una unión que debe retirarse y definir una propiedad CLR ordinaria en lugar de datos.

Así que cada vez que se crea una clase que deriva de bindableObject, una de las primeras piezas de código que debería estar escribiendo en esa clase comienza "BindableProperty sólo lectura estática pública" -tal vez la secuencia más característico de cuatro palabras en toda la programación Xamarin.Forms.

Definir las propiedades enlazables

Supongamos que desea una mejorada Etiqueta la clase que le permite especificar tamaños de fuente en unidades de puntos. Vamos a llamar a esta clase altLabel para "alternativo Etiqueta. "Se deriva de Etiqueta e incluye una nueva propiedad llamada PointSize.

Debería PointSize ser respaldado por una propiedad que puede vincularse? ¡Por supuesto! (A pesar de las ventajas reales de hacerlo no se demostrarán hasta los próximos capítulos.)

El código de sólo altLabel clase está incluido en el **Xamarin.FormsBook.Toolkit** biblioteca, por lo que es accesible para múltiples aplicaciones. El nuevo PointSize la propiedad se lleva a cabo con una BindableProperty objeto nombrado PointSizeProperty y una propiedad CLR nombrado PointSize que las referencias PointSizeProperty:

```
clase pública altLabel : Etiqueta
{
    sólo lectura estática pública BindableProperty PointSizeProperty ...;
    ...
    pública doble PointSize
    {
        conjunto {EstablecerValor (PointSizeProperty, valor);}
        obtener {regreso (doble) GetValue (PointSizeProperty);}
    }
    ...
}
```

Tanto el campo como la definición de la propiedad deben ser públicas.

Porque PointSizeProperty Se define como estático y solo lectura, se debe asignar ya sea en un constructor estático o hacia la derecha en la definición de campo, después de lo cual no se puede cambiar. Generalmente, una BindableProperty objeto se le asigna en la definición de campo mediante el uso de la estática BindableProperty.Create método. Se requieren cuatro argumentos (que se muestra aquí con los nombres de los argumentos):

- nombre de la propiedad El nombre de texto de la propiedad (en este caso "PointSize")
- returnType El tipo de la propiedad (una doble en este ejemplo)
- declaringType El tipo de la clase que define la propiedad (altLabel)
- valor por defecto Un valor por defecto (digamos 8 puntos)

Los argumentos segundo y tercero se definen generalmente con tipo de expresiones. Aquí está la instrucción de asignación de estos cuatro argumentos que se pasan a BindableProperty.Create:

```

público clase altLabel : Etiqueta
{
    público estático solo lectura BindableProperty PointSizeProperty =
        BindableProperty .Crear( "PointSize" ,
            tipo de ( doble ), // nombre de la propiedad
            tipo de ( altLabel ), // returnType
            8,0, // declaringType
            ...); // valor por defecto
    ...
}

```

Observe que el valor predeterminado se especifica como 8,0 en lugar de sólo 8. Debido a que BindableProperty.Create está diseñado para manejar propiedades de cualquier tipo, el valor por defecto parámetro se define como objeto. Cuando el compilador de C# se encuentra a sólo 8 en ese argumento, se asume que el 8 es una Entidad y no un tipo. El problema no será revelado hasta que el tiempo de ejecución, sin embargo, cuando el BindableProperty.Create método se espera que el valor por defecto a ser de tipo doble y responder al elevar una TypeInitializationException.

Debe ser explícita sobre el tipo del valor que está especificando como predeterminado. No hacerlo es un error muy común en la definición de propiedades enlazables. UN *muy* error común.

BindableProperty.Create También cuenta con seis argumentos opcionales. Aquí están los nombres de los argumentos y su propósito:

- defaultBindingMode Usado en conexión con el enlace de datos
- validateValue Una devolución de llamada para comprobar si hay un valor válido
- PropertyChanged Una devolución de llamada para indicar cuando la propiedad ha cambiado
- propertyChanging Una devolución de llamada para indicar cuando la propiedad está a punto de cambiar
- coerceValue Una devolución de llamada para obligar a un valor de ajuste a otro valor (por ejemplo, para restringir los valores a una gama)

- defaultValueCreator Una devolución de llamada para crear un valor predeterminado. Esto se utiliza generalmente a una instancia de un objeto por defecto que no puede ser compartido entre todas las instancias de la clase; por ejemplo, una colección de objetos tales como Lista o Diccionario.

No lleve a cabo ninguna validación, la coacción o la propiedad que cambió el manejo de la propiedad CLR. La propiedad CLR debe limitarse a Valor ajustado y GetValue llamadas. Todo lo demás debe hacerse en las devoluciones de llamada proporcionados por la infraestructura de propiedad enlazable.

Es muy raro que una llamada en particular a BindableProperty.Create necesitaría todos estos argumentos opcionales. Por esa razón, estos argumentos opcionales se indican comúnmente con la función argumento con nombre introducido en C# 4.0. Para especificar un argumento opcional en particular, utilice el nombre del argumento seguido de dos puntos. Por ejemplo:

```

clase pública altLabel : Etiqueta
{
    sólo lectura estática pública BindableProperty PointSizeProperty =
        BindableProperty .Create("PointSize", // nombre de la propiedad
            tipo de ( doble ), // returnType
            tipo de ( altLabel ), // declaringType
            8,0, // valor por defecto
            PropertyChanged: OnPointSizeChanged);
    ...
}

```

Sin duda, PropertyChanged es el más importante de los argumentos opcionales debido a que la clase utiliza esta devolución de llamada para ser notificado cuando cambia la propiedad, ya sea directamente desde una llamada a Conjunto-Valor o a través de la propiedad de CLR.

En este ejemplo, el controlador de la propiedad se llama cambiado OnPointSizeChanged. Se va a llamar sólo cuando la propiedad realmente cambia y no cuando se trata simplemente establece en el mismo valor. Sin embargo, debido a que OnPointSizeChanged se hace referencia a partir de un campo estático, el método en sí también debe ser estática. Esto es lo que parece:

```

clase pública altLabel : Etiqueta
{
    ...
    ...
    hoyo estatico OnPointSizeChanged ( bindableObject enlazable, objeto valor antiguo, objeto nuevo valor)
    {
        ...
    }
    ...
}

```

Esto parece un poco extraño. Podríamos tener múltiples altLabel instancias en un programa, sin embargo, siempre que la PointSize cambios de propiedad en cualquiera de estos casos, se llama a este mismo método estático. ¿Cómo funciona el método de saber exactamente qué altLabel instancia ha cambiado?

El método puede decir cuál de propiedad de instancia ha cambiado porque esa instancia es siempre el primer argumento al manejador de propiedad cambiado. A pesar de que el primer argumento se define como una BindableObject, en este caso, en realidad es de tipo altLabel e indica que altLabel la propiedad de instancia ha cambiado. Esto significa que se puede lanzar con seguridad el primer argumento a una altLabel ejemplo:

```

hoyo estatico OnPointSizeChanged ( bindableObject enlazable, objeto valor antiguo, objeto nuevo valor)
{
    altLabel altLabel = ( altLabel ) Enlazable;
    ...
}

```

A continuación, puede hacer referencia a cualquier cosa en el caso particular de altLabel cuya propiedad ha cambiado. Los argumentos segundo y tercero son realmente de tipo doble para este ejemplo e indicar el valor anterior y el nuevo valor.

A menudo es conveniente para este método estático para llamar a un método de instancia con los argumentos convertidos a sus tipos reales:

```

pública clase AltLabel : Etiqueta
{
    ...
    hoyo estatico OnPointSizeChanged ( bindableObject enlazable, objeto valor antiguo, objeto nuevo valor)
    {
        (( altLabel ) Enlazable) .OnPointSizeChanged (( doble )valor antiguo, ( doble )nuevo valor);
    }

    vacío OnPointSizeChanged ( doble valor antiguo, doble nuevo valor)
    {
        ...
    }
}

```

El método de instancia continuación, puede hacer uso de cualquier propiedades de la instancia o métodos de la clase base subyacente como lo haría normalmente.

Para esta clase, este OnPointSizeChanged método necesita para establecer el Tamaño de fuente propiedad basada en el nuevo tamaño de punto y un factor de conversión. Además, el constructor debe inicializar el Fuente-tamaño propiedad basada en el valor por defecto PointSize valor. Esto se hace a través de un simple SetLabelFontSize método. Aquí está la clase final completo:

```

pública clase AltLabel : Etiqueta
{
    pública estatico solo lectura BindableProperty PointSizeProperty =
        BindableProperty .Crear( "PointSize" ,
            BindableProperty .Crear( "PointSize" , // nombre de la propiedad
                tipo de ( doble ), // returnType
                tipo de ( AltLabel ), // declaringType
                8,0, // valor por defecto
                PropertyChanged: OnPointSizeChanged);

    pública AltLabel ()
    {
        SetLabelFontSize (( doble ) PointSizeProperty.DefaultValue);
    }

    pública doble PointSize
    {
        conjunto {EstablecerValor (PointSizeProperty, valor); }
        obtener { regreso ( doble ) GetValue (PointSizeProperty); }
    }

    hoyo estatico OnPointSizeChanged ( bindableObject enlazable, objeto valor antiguo, objeto nuevo valor)
    {
        (( altLabel ) Enlazable) .OnPointSizeChanged (( doble )valor antiguo, ( doble )nuevo valor);
    }

    vacío OnPointSizeChanged ( doble valor antiguo, doble nuevo valor)
    {
        SetLabelFontSize (newValue);
    }

    vacío SetLabelFontSize ( doble pointsize)
}

```

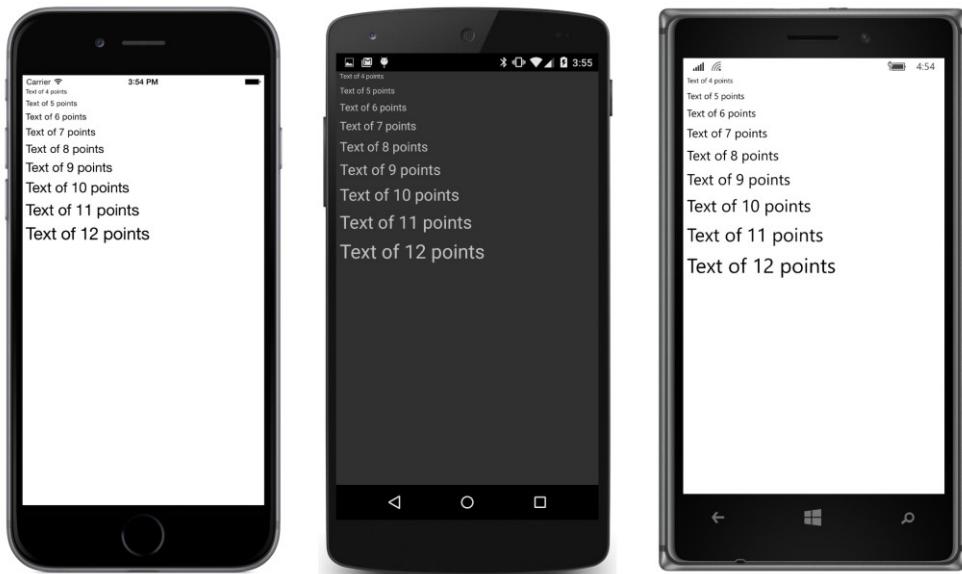
```
{  
    FontSize = 160 * pointsize / 72;  
}  
}
```

También es posible para la instancia **OnPointSizeChanged** propiedad para acceder a la **PointSize** la propiedad directamente en lugar de uso nuevo valor. En el momento en el controlador de la propiedad se llama cambiado, el valor de la propiedad subyacente ya ha sido cambiado. Sin embargo, usted no tiene acceso directo a ese valor subyacente, como lo hace cuando un campo privado respalda una propiedad CLR. Ese valor subyacente es privado a **BindableObject** y accesible sólo a través de la **GetValue** llamada.

Por supuesto, nada impide que el código que está utilizando **altLabel** desde el establecimiento de la Tamaño de fuente propiedad y anular la **PointSize** el establecimiento, pero esperemos que dicho código es consciente de ello. Aquí hay un código que es, un programa llamado **PointSizedText**, que utiliza **altLabel** para mostrar tamaños de punto de 4 a 12:

```
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "  
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "  
    xmlns: kit de herramientas =  
        "clr-espacio de nombres: Xamarin.FormsBook.Toolkit; montaje = Xamarin.FormsBook.Toolkit "  
    x: Class = "PointSizedText.PointSizedTextPage " >  
  
< ContentPage.Padding >  
    < OnPlatform x: TypeArguments = " Espesor "  
        iOS = " 5, 20, 0, 0 "  
        Androide = " 5, 0, 0, 0 "  
        WinPhone = " 5, 0, 0, 0 " />  
  
</ ContentPage.Padding >  
  
< StackLayout x: Nombre = " stackLayout " >  
    < kit de herramientas: altLabel Texto = " Texto de 4 puntos " PointSize = " 4 " />  
    < kit de herramientas: altLabel Texto = " Texto de 5 puntos " PointSize = " 5 " />  
    < kit de herramientas: altLabel Texto = " Texto de 6 puntos " PointSize = " 6 " />  
    < kit de herramientas: altLabel Texto = " Texto de 7 puntos " PointSize = " 7 " />  
    < kit de herramientas: altLabel Texto = " Texto de 8 puntos " PointSize = " 8 " />  
    < kit de herramientas: altLabel Texto = " Texto de 9 puntos " PointSize = " 9 " />  
    < kit de herramientas: altLabel Texto = " Texto de 10 puntos " PointSize = " 10 " />  
    < kit de herramientas: altLabel Texto = " Texto de 11 puntos " PointSize = " 11 " />  
    < kit de herramientas: altLabel Texto = " Texto de 12 puntos " PointSize = " 12 " />  
  
</ StackLayout >  
</ Pagina de contenido >
```

Y aquí están las capturas de pantalla:



La propiedad enlazable de sólo lectura

Supongamos que estás trabajando con una aplicación en la que es conveniente conocer el número de palabras en el texto que se muestra por una

Etiqueta elemento. Tal vez le gustaría construir esa instalación a la derecha en una clase que deriva de Etiqueta. Vamos a llamar a esta nueva clase CountedLabel.

Por ahora, su primer pensamiento debe ser definir una BindableProperty objeto nombrado El recuento de palabras-Propiedad y una correspondiente propiedad CLR nombrado El recuento de palabras.

Pero, un momento: Sólo tiene sentido para este El recuento de palabras propiedad que se establece dentro de la CountedLabel clase. Eso significa que el El recuento de palabras CLR propiedad no debe tener un público conjunto descriptor de acceso. Se debe definir de esta manera:

```
public int El recuento de palabras
{
    conjunto privado {EstablecerValor (WordCountProperty, valor);}
    obtener {regreso (doble) GetValue (WordCountProperty);}
}
```

los obtener descriptor de acceso sigue siendo pública, pero la conjunto descriptor de acceso es privado. ¿Es suficiente?

No exactamente. A pesar de lo privado conjunto descriptor de acceso en la propiedad CLR, código externo a CountedLabel todavía se puede llamar Valor ajustado con el CountedLabel.WordCountProperty inmueble objeto enlazable. Ese tipo de configuración de la propiedad debe prohibirse también. Pero, ¿cómo puede funcionar eso si el WordCountProperty objeto es público?

La solución es hacer una *solo lectura* propiedad que puede vincularse mediante el uso de la BindableProperty.Create-Solo lectura método. los

Xamarin.Forms sí API define varias propiedades; por ejemplo de sólo lectura enlazable, la Anchura y Altura propiedades definidas por VisualElement.

He aquí cómo usted puede hacer una propia:

El primer paso es llamar BindableProperty.CreateReadOnly con los mismos argumentos como para BindableProperty.Create. sin embargo, el CreateReadOnly método devuelve un objeto de BindablePropertyKey más bien que BindableProperty. Definir este objeto como estático y solo lectura, como con el BindableProperty, pero hacen que sea privada de la clase:

```
clase pública CountedLabel : Etiqueta
{
    estatico solo lectura BindablePropertyKey WordCountKey =
        BindableProperty .CreateReadOnly ("El recuento de palabras",
            tipo de ( Ent ),
            tipo de ( CountedLabel ),
            0);
```

// nombre de la propiedad
// returnType
// declaringType
// valor por defecto

...

}

No creo que de esta BindablePropertyKey objeto como clave de cifrado o algo por el estilo. Es mucho más simple, en realidad sólo un objeto que es privado para la clase.

El segundo paso es hacer una pública BindableProperty objeto mediante el BindableProperty propiedad de la BindablePropertyKey:

```
clase pública CountedLabel : Etiqueta
{
    ...
    sólo lectura estatica pública BindableProperty WordCountProperty = WordCountKey.BindableProperty;
    ...
}
```

Esta BindableProperty objeto es pública, pero es un tipo especial de BindableProperty: No se puede utilizar en una Valor ajustado llamada. Si intenta hacerlo, se producirá un InvalidOperationException.

Sin embargo, hay una sobrecarga de la Valor ajustado método que acepta una BindablePropertyKey objeto. el CLR conjunto descriptor de acceso puede llamar Valor ajustado el uso de este objeto, pero esto conjunto descriptor de acceso debe ser privada para evitar que la propiedad de ser establecido fuera de la clase:

```
clase pública CountedLabel : Etiqueta
{
    ...
    public int El recuento de palabras
    {
        conjunto privado {EstablecerValor (WordCountKey, valor);}
        obtener {regreso ( Ent ) GetValue (WordCountProperty);}
    }
    ...
}
```

los El recuento de palabras propiedad ahora se puede ajustar desde el interior de la CountedLabel clase. Pero cuando debe el

clase se establece? Esta CountedLabel clase se deriva de Etiqueta, pero es necesario para detectar cuando el Texto la propiedad ha cambiado de modo que pueda contar hasta las palabras.

Hace Etiqueta tener un TextChanged ¿evento? No, no lo hace. Sin embargo, bindableObject implementa el INotifyPropertyChanged interfaz.
Se trata de una interfaz muy importante .NET, particularmente para aplicaciones que implementan la arquitectura Model-View-ViewModel (MVVM). En el capítulo 18 verá cómo usarlo en sus propias clases de datos.

los INotifyPropertyChanged interfaz se define en el System.ComponentModel espacio de nombres de este modo:

```
p\xedblico interfaz INotifyPropertyChanged
{
    evento PropertyChangedEventHandler PropertyChanged;
}
```

Cada clase que deriva de bindableObject se dispara automáticamente este PropertyChanged evento cada vez que cualquier propiedad respaldado por una BindableProperty cambia. Los PropertyChangedEventArgs objeto que acompaña este evento incluye una propiedad denominada Nombre de la propiedad de tipo cuerda que identifica la propiedad que ha cambiado.

As\xed que todo lo que es necesario es que CountedLabel adjuntar un controlador para el PropertyChanged evento y cheque por un nombre de propiedad de "texto". Desde all\xed se puede usar cualquier t\xecnica que quiere para calcular un recuento de palabras. El completo CountedLabel clase utiliza una funci\xon lambda en el Propiedad-cambiado evento. Las llamadas de controlador Divisi\xon para romper la cadena en palabras y ver cu\xf3ntos resultan piezas. Los Divisi\xon m\xetodo divide el texto en funci\xon de los espacios, guiones y guiones largos (Unicode \ u2014):

```
p\xedblico clase CountedLabel : Etiqueta
{
    est\u00e1tico solo lectura BindablePropertyKey WordCountKey =
        BindableProperty.CreateReadOnly ("El recuento de palabras",           // nombre de la propiedad
                                         tipo de (En t),                  // returnType
                                         tipo de (CountedLabel),          // declaringType
                                         0);                                // valor por defecto

    p\xedblico est\u00e1tico solo lectura BindableProperty WordCountProperty = WordCountKey.BindableProperty;

    p\xedblico CountedLabel ()
    {
        // Establecer la propiedad WordCount cuando cambia la propiedad de texto.
        PropertyChanged += (objeto remitente, PropertyChangedEventArgs args) =>
        {
            Si (Args.PropertyName == "Texto")
            {
                Si (Cuerda.IsNullOrEmpty (texto))
                {
                    WordCount = 0;
                }
                m\u00e1s
                {
                    WordCount = Text.Split (" ,\u2014 ,\u2014 U2014").Longitud;
                }
            }
        };
    }
}
```

```

        }
    }

}

public int El recuento de palabras
{
    conjunto privado {EstablecerValor (WordCountKey, valor);}
    obtener {regreso (En t) GetValue (WordCountProperty);}
}
}

```

La clase incluye una utilizando Directiva para la System.ComponentModel espacio de nombres para el PropiedadChangedEventArgs argumento al manejador. Cuidado con: Xamarin.Forms define una clase denominada ApuntalarertyChangingEventArgs (tiempo presente). Eso no es lo que desea para el PropertyChanged entrenador de animales. Usted quiere PropertyChanged (pasado).

Debido a esta llamada de la División método divide el texto en blanco personajes, guiones y guiones largos, es posible suponer que CountedLabel se demostrará con el texto que contiene algunos guiones y guiones largos. Esto es verdad. los **BaskervillesCount** programa es una variación de la **Baskerville** programa desde el capítulo 3, pero aquí el párrafo de texto se muestra con una CountedLabel, y un habitual

Etiqueta se incluye para mostrar el número de palabras:

```

< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    xmlns: kit de herramientas =
        "clr-espacio de nombres: Xamarin.FormsBook.Toolkit; montaje = Xamarin.FormsBook.Toolkit "
    x: Class = " BaskervillesCount.BaskervillesCountPage "
    Relleno = " 5, 0 " >

    < StackLayout >
        < kit de herramientas: CountedLabel x: Nombre = " countedLabel "
            VerticalOptions = " CenterAndExpand "
            Texto =
                " Sherlock Holmes, que era por lo general muy tarde en
                Por las mañanas, salvo a aquellos que no infrecuentes
                ocasiones en las que estuvo despierto toda la noche, estaba sentado en el
                la mesa del desayuno. Me paré sobre la solera-alfombra
                y recogí el bastón que nuestro visitante había dejado
                detrás de él la noche anterior. Era una multa, de espesos
                pedazo de madera, con bulbo-dirigió, del tipo que
                que se conoce como una & # X201C; abogado de Penang, & # X201D; Sólo
                bajo la cabeza era una banda de plata amplia, casi una
                pulgada de ancho, & # X201C; Para James Mortimer, Cruz Roja de Myanmar,
                de sus amigos de la CCH, & # X201D; fue grabado
                sobre ella, con la fecha & # X201C; 1884. & # X201D; Era
                Sólo un palo como la antigua familia
                practicante utiliza para llevar & # X2014; digno, sólido,
                y tranquilizador. *>

    < Etiqueta x: Nombre = " wordCountLabel "
        Texto = " ??? "

```

```

    Tamaño de fuente = "Grande"
    VerticalOptions = "CenterAndExpand"
    HorizontalOptions = "Centrar" />

</StackLayout>
</Pagina de contenido>

```

Eso regulares Etiqueta se encuentra en el archivo de código subyacente:

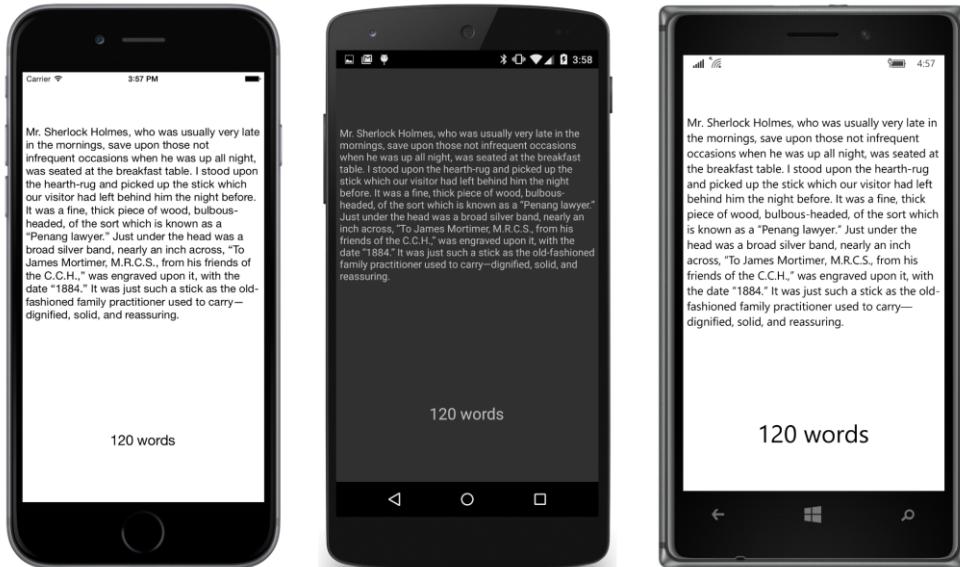
```

pública clase parcial BaskervillesCountPage : Pagina de contenido
{
    pública BaskervillesCountPage ()
    {
        InitializeComponent ();

        En t WordCount = countedLabel.WordCount;
        wordCountLabel.Text = WordCount + " palabras";
    }
}

```

El recuento de palabras que se calcula se basa en la suposición de que todos los guiones en el texto dos palabras separadas y que "hogar-alfombra" y "-bulbosa titulada" deben contarse como dos palabras cada uno. Eso no es siempre cierto, por supuesto, pero el recuento de palabras no son tan simples como algorítmicamente este código podría implicar:



¿Cómo sería el programa se estructurará si el texto cambia dinámicamente mientras el programa se ejecuta? En ese caso, sería necesario actualizar el número de palabras cada vez que el El recuento de palabras propiedad de la CountedLabel Objeto cambiado. Se podría adjuntar una PropertyChanged manejador en el Contar-edLabel objeto y comprobar la propiedad denominada "WordCount".

Sin embargo, tenga cuidado si se intenta establecer un manejador de tal evento desde XAML, por ejemplo, así:

```
<kit de herramientas: CountedLabel x: Nombre = " countedLabel "
    VerticalOptions = " CenterAndExpand "
    PropertyChanged = " OnCountedLabelPropertyChanged "
    Texto = "... ">
```

Es probable que desee para codificar el controlador de eventos en el archivo de código subyacente de esta manera:

```
vacio OnCountedLabelPropertyChanged ( objeto remitente,
                                         PropertyChangedEventArgs args)
{
    wordCountLabel.Text = countedLabel.WordCount + " palabras";
}
```

Ese controlador se disparará cuando el Texto propiedad se establece por el analizador XAML, pero el controlador de eventos está tratando de establecer el Texto propiedad de la segunda Etiqueta, que no ha sido todavía instancia, lo que significa que el wordCountLabel campo de fotografías está ajustado a nulo. Este es un tema que se van a plantear de nuevo en el capítulo 15 cuando se trabaja con controles interactivos, pero será más o menos resuelto cuando trabajamos con el enlace de datos en el capítulo 16.

Hay otra variación de una propiedad que puede vincularse viene en Capítulo 14 en el AbsoluteLayout: este es el *adjunta propiedad enlazable*, y es muy útil en la aplicación de ciertos tipos de diseños, como también descubrirá en el capítulo 26, "diseños personalizados."

Mientras tanto, vamos a ver una de las aplicaciones más importantes de las propiedades enlazables: estilos.

capítulo 12

estilos

Xamarin.Forms aplicaciones a menudo contienen múltiples elementos con valores de propiedades idénticas. Por ejemplo, es posible tener varios botones con los mismos colores, tamaños de fuente y opciones de diseño. En el código, puede asignar propiedades idénticas a varios botones en un bucle, pero bucles no está disponible en XAML. Si se quiere evitar una gran cantidad de marcas repetitivo, se requiere otra solución.

La solución es el Estilo clase, que es una colección de valores de propiedades consolidadas en un objeto conveniente. Se puede establecer una Estilo oponerse a la Estilo bienes de cualquier clase que deriva de `VisualElement`. En general, vamos a aplicar la misma Estilo objeto de múltiples elementos, y el estilo es compartida entre estos elementos.

los Estilo es la herramienta principal para dar elementos visuales una apariencia coherente en sus aplicaciones Xamarin.Forms. Estilos ayudan a reducir el margen de beneficio repetitivo en archivos XAML y permite que las aplicaciones sean cambiados y se mantienen más fácilmente.

Estilos fueron diseñados principalmente con XAML en mente, y que probablemente no se han inventado en un entorno de sólo código. Sin embargo, se verá en este capítulo cómo definir y utilizar estilos en el código y cómo combinar código y marcadores para cambiar el estilo del programa dinámicamente en tiempo de ejecución.

El estilo básico

En el capítulo 10, "extensiones de marcado XAML", que vieron un trío de botones que contenían una gran cantidad de marcas idénticas. Aquí están de nuevo:

```
< StackLayout >
    < Botón Texto = "Carpe Diem" >
        HorizontalOptions = "Centrar"
        VerticalOptions = "CenterAndExpand"
        Ancho del borde = "3"
        Color de texto = "rojo"
        Tamaño de fuente = "Grande" >
    < Button.BackgroundColor >
        < OnPlatform x: TypeArguments = "Color" >
            Androide = "# 404040" />
        </ Button.BackgroundColor >
    < Button.BorderColor >
        < OnPlatform x: TypeArguments = "Color" >
            Androide = "Blanco"
            WinPhone = "Negro" />
        </ Button.BorderColor >
    </ Botón >
```

```

< Botón Texto = " Sapere aude "
    HorizontalOptions = " Centrar "
    VerticalOptions = " CenterAndExpand "
    Ancho del borde = " 3 "
    Color de texto = " rojo "
    Tamaño de fuente = " Grande " >
< Button.BackgroundColor >
    < OnPlatform x: TypeArguments = " Color "
        Androide = "#404040" />
</ Button.BackgroundColor >
< Button.BorderColor >
    < OnPlatform x: TypeArguments = " Color "
        Androide = " Blanco "
        WinPhone = " Negro " />
</ Button.BorderColor >
</ Botón >

< Botón Texto = " faciendo discos "
    HorizontalOptions = " Centrar "
    VerticalOptions = " CenterAndExpand "
    Ancho del borde = " 3 "
    Color de texto = " rojo "
    Tamaño de fuente = " Grande " >
< Button.BackgroundColor >
    < OnPlatform x: TypeArguments = " Color "
        Androide = "#404040" />
</ Button.BackgroundColor >
< Button.BorderColor >
    < OnPlatform x: TypeArguments = " Color "
        Androide = " Blanco "
        WinPhone = " Negro " />
</ Button.BorderColor >
</ Botón >
</ StackLayout >

```

Con la excepción de la Texto propiedad, los tres botones tienen los mismos valores de propiedades.

Una solución parcial a este marcado repetitivo implica la definición de los valores de propiedad en un diccionario de recursos y hacer referencia a ellas con el StaticResource extensión de marcado. Como se vio en el

El intercambio de recursos proyecto en el capítulo 10, esta técnica no reduce el margen de beneficio mayor, pero sí a consolidar los valores en un solo lugar.

Para reducir el mayor margen de beneficio, se necesita un Estilo. UN Estilo objeto casi siempre se define en una ResourceDictionary. En general, usted comenzará con una recursos sección en la parte superior de la página:

```

< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " BasicStyle.BasicStylePage " >

    < ContentPage.Resources >
        < ResourceDictionary >
            ...
        </ ResourceDictionary >

```

```
</ ContentPage.Resources >  
...  
</ Pagina de contenido >  
  
una instancia de un Estilo con las etiquetas de inicio y fin separadas:  
  
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "  
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "  
    x: Class = " BasicStyle.BasicStylePage " >  
  
    < ContentPage.Resources >  
        < ResourceDictionary >  
            < Estilo x: Key = " buttonStyle " Tipo de objetivo = " Botón " >  
                ...  
            </ Estilo >  
        </ ResourceDictionary >  
    </ ContentPage.Resources >  
...  
</ Pagina de contenido >
```

Porque el Estilo es un objeto en una ResourceDictionary, se necesita de una x: Key atribuir a darle un descriptiva clave de diccionario. También debe establecer la Tipo de objetivo propiedad. Este es el tipo del elemento visual de que el estilo está diseñado para que, en este caso es Botón.

Como se verá en la siguiente sección de este capítulo, también se puede definir una Estilo en el código, en cuyo caso el Estilo constructor requiere un objeto de tipo Tipo Para el Tipo de objetivo propiedad. Los Tipo de objetivo propiedad no tiene un público conjunto accesos; por lo tanto, la Tipo de objetivo propiedad no se puede cambiar después de la Estilo es creado.

Estilo también define otra importante conseguir establecimiento sólo nombre setters de tipo IList<Setter>, que es una colección de Setter objetos. Cada Setter es responsable de la definición de un valor de la propiedad en el estilo. Los Setter clase define sólo dos propiedades:

- Propiedad de tipo BindableProperty
- Valor de tipo Objeto

Las propiedades establecidas en el Estilo deben ser respaldados por las propiedades enlazables! Pero cuando se establece el Propiedad propiedad en XAML, no utilice todo el nombre de la propiedad enlazable completo. Sólo especifique el nombre del texto, que es el mismo que el nombre de la propiedad CLR relacionada. He aquí un ejemplo:

```
< Setter Propiedad = " HorizontalOptions " Valor = " Centrar " />
```

El analizador XAML utiliza el familiar TypeConverter clases cuando el análisis sintáctico Valor la configuración de éstos Setter casos, por lo que se pueden utilizar los mismos valores de propiedades que se utilizan normalmente.

Setters es la propiedad de contenido Estilo, por lo que no es necesario el Style.Setters etiquetas para añadir Setter se opone a la Estilo:

```
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "  
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
```

```

x: Class = " BasicStyle.BasicStylePage " >

< ContentPage.Resources >
  < ResourceDictionary >
    < Estilo x: Key = " buttonStyle " Tipo de objetivo = " Botón " >
      < Setter Propiedad = " HorizontalOptions " Valor = " Centrar " />
      < Setter Propiedad = " VerticalOptions " Valor = " CenterAndExpand " />
      < Setter Propiedad = " Ancho del borde " Valor = " 3 " />
      < Setter Propiedad = " Color de texto " Valor = " rojo " />
      < Setter Propiedad = " Tamaño de fuente " Valor = " Grande " />
    ...
  </ Estilo >
</ ResourceDictionary >
</ ContentPage.Resources >
...
</ Página de contenido >

```

Dos más Setter Se requieren los objetos de Color de fondo y Color del borde. estos implican

OnPlatform y podría parecer a primera vista que es imposible de expresar en el marcado. Sin embargo, es posible expresar el Valor propiedad de Setter como un elemento de propiedad, con el OnPlatform markup entre las etiquetas de los elementos de propiedad:

```

< Setter Propiedad = " Color de fondo " >
  < Setter.Value >
    < OnPlatform x: TypeArguments = " Color "
      Androide = "# 404040 " />
  </ Setter.Value >
</ Setter >
< Setter Propiedad = " Color del borde " >
  < Setter.Value >
    < OnPlatform x: TypeArguments = " Color "
      Androide = " Blanco "
      WinPhone = " Negro " />
  </ Setter.Value >
</ Setter >

```

El paso final es para establecer este Estilo oponerse a la Estilo propiedad de cada Botón. Utilizar el familiar

StaticResource extensión de marcado para hacer referencia a la clave de diccionario. Aquí está el archivo XAML completa en el **BasicStyle** proyecto:

```

< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
  xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
  x: Class = " BasicStyle.BasicStylePage " >

< ContentPage.Resources >
  < ResourceDictionary >
    < Estilo x: Key = " buttonStyle " Tipo de objetivo = " Botón " >
      < Setter Propiedad = " HorizontalOptions " Valor = " Centrar " />
      < Setter Propiedad = " VerticalOptions " Valor = " CenterAndExpand " />
      < Setter Propiedad = " Ancho del borde " Valor = " 3 " />
      < Setter Propiedad = " Color de texto " Valor = " rojo " />
      < Setter Propiedad = " Tamaño de fuente " Valor = " Grande " />
      < Setter Propiedad = " Color de fondo " >

```

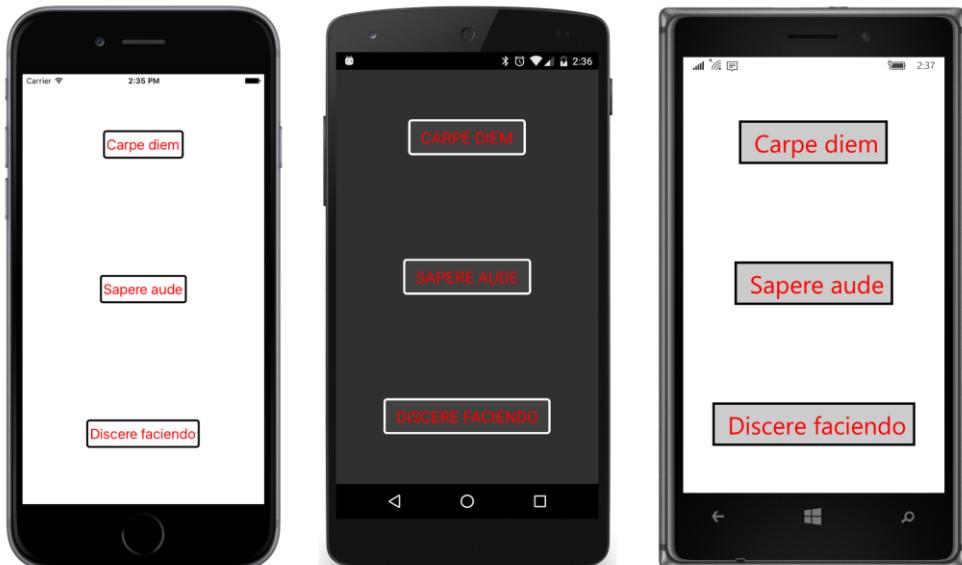
```
<Setter.Value>
<OnPlatform x:TypeArguments = "Color"
    Androide = "#404040" />
</Setter.Value>
</Setter>
<Setter Propiedad = "Color del borde">
<Setter.Value>
<OnPlatform x:TypeArguments = "Color"
    Androide = "Blanco"
    WinPhone = "Negro" />
</Setter.Value>
</Setter>
</Estilo>
</ResourceDictionary>
</ContentPage.Resources>

<StackLayout>
<Botón Texto = "Carpe Diem"
    Estilo = "@{StaticResource buttonStyle}" />

<Botón Texto = "Sapere audē"
    Estilo = "@{StaticResource buttonStyle}" />

<Botón Texto = "faciendo discere"
    Estilo = "@{StaticResource buttonStyle}" />
</StackLayout>
</Pagina de contenido>
```

Ahora todos estos valores de propiedades se encuentran en una Estilo objeto que se comparte entre múltiples Botón elementos:



Los efectos visuales son los mismos que los de la **El intercambio de recursos** programa en el capítulo 10, pero el margen de beneficio es mucho más concisa.

Incluso después de trabajar con Estilo objetos en el marcado, es fácil estar desconcertado con un poco manejable **Valor propiedad**. Supongamos que desea definir una Setter Para el Color de texto utilizando el **Color.FromHsla** método estático. Se puede definir un color mediante el uso de la **x: FactoryMethod** atribuye, pero ¿cómo es posible fijar una porción tan difícil de manejar de marcado para el Valor propiedad de la Setter ¿objeto? Como se vio anteriormente, la solución es casi siempre sintaxis de la propiedad de elementos:

```
<ResourceDictionary>
  <Estilo x:Key = "buttonStyle" Tipo de objetivo = "Botón" >
    ...
    <Setter Propiedad = "Color de texto" >
      <Setter.Value>
        <Color x:FactoryMethod = "FromHsla" >
          <x:Argumentos>
            <x:Doble> 0.83 </x:Doble>
            <x:Doble> 1 </x:Doble>
            <x:Doble> 0.75 </x:Doble>
            <x:Doble> 1 </x:Doble>
          </x:Argumentos>
        </Color>
      </Setter.Value>
    </Setter>
    ...
  </Estilo>
</ResourceDictionary>
```

Aquí hay otra manera de hacerlo: Definir el Color valor como punto separado en el diccionario de recursos, y luego usar **StaticResource** para establecerlo en el Valor propiedad de la Setter:

```
<ResourceDictionary>
  <Color x:Key = "btnTextColor" 
    x:FactoryMethod = "FromHsla" >
    <x:Argumentos>
      <x:Doble> 0.83 </x:Doble>
      <x:Doble> 1 </x:Doble>
      <x:Doble> 0.75 </x:Doble>
      <x:Doble> 1 </x:Doble>
    </x:Argumentos>
  </Color>
  <Estilo x:Key = "buttonStyle" Tipo de objetivo = "Botón" >
    ...
    <Setter Propiedad = "Color de texto" Valor = "StaticResource btnTextColor" />
    ...
  </Estilo>
</ResourceDictionary>
```

Esta es una buena técnica si va a compartir la misma Color valor entre varios estilos o varios emisores.

Puede anular un valor de propiedad de una Estilo estableciendo una propiedad directamente en el elemento visual. Observe que el segundo Botón tiene su Color de texto propiedad establecida en Granate:

```
< StackLayout >
    < Botón Texto = "Carpe Diem"
        Estilo = "{} StaticResource buttonStyle" />

    < Botón Texto = "Sapere aude"
        Color de texto = "Granate"
        Estilo = "{} StaticResource buttonStyle" />

    < Botón Texto = "faciendo discere"
        Estilo = "{} StaticResource buttonStyle" />
</ StackLayout >
```

el centro Botón tendrá el texto marrón, mientras que los otros dos botones consiguen su Color de texto la configuración de la Estilo. Una propiedad directamente situado en el elemento visual a veces se llama una *Configuración local* o una *Ajuste manual*, y siempre prevalece sobre la propiedad de la configuración de Estilo.

los Estilo objeto en el **BasicStyle** programa es compartida entre los tres botones. El intercambio de estilos tiene una implicación importante para la Setter objetos. Cualquier objeto establecido en el Valor propiedad de una Setter debe ser compatibles. No trate de hacer algo como esto:

```
<! - XAML no válido! ->
< Estilo x: Key = "de marco" Tipo de objetivo = "Marco" >
    < Setter Propiedad = "OutlineColor" Valor = "Acento" />
    < Setter Propiedad = "Contenido" >
        < Setter.Value >
            < Etiqueta Texto = "Texto en un marco" />
        </ Setter.Value >
    </ Setter >
</ Estilo >
```

Este XAML no funciona por dos razones: Contenido No está respaldado por una BindableProperty y por lo tanto no puede ser utilizado en una Setter. Pero la intención obvia aquí es que cada Marco o por lo menos cada Marco sobre la que se aplica a este estilo conseguir que la misma Etiqueta oponerse como contenido. Un único Etiqueta objeto no puede aparecer en varios lugares de la página. Una mejor manera de hacer algo como esto es para derivar una clase de Marco y establecer una Etiqueta como el Contenido propiedad, o para derivar una clase de ContentView que incluye una Marco y Etiqueta.

Es posible que desee utilizar un estilo para configurar un controlador de eventos para un evento como Se hace clic. Eso sería útil y conveniente, pero no es compatible. Los controladores de eventos se deben establecer en los propios elementos. (Sin embargo, el Estilo clase de objetos de soporte hace llamadas *disparadores*, que puede responder a eventos o cambios en las propiedades. Los disparadores se tratan en el capítulo 23, "disparadores y comportamientos.")

No se puede establecer la GestureRecognizers propiedad en un estilo. Eso sería útil también, pero GestureRecognizers No está respaldado por una propiedad enlazable.

Si una propiedad enlazable es un tipo de referencia, y si el valor por defecto es nulo, se puede utilizar un estilo para establecer la propiedad de un no nulo objeto. Pero también puede ser que desee omiso de esa preferencia estilo con un local

configuración que establece la propiedad de nuevo a nulo. Puede establecer una propiedad de nulo en XAML con el {X: Null} extensión de marcado.

Estilos de código

Aunque los estilos son en su mayoría definidos y utilizados en XAML, usted debe saber lo que parecen cuando se define y utiliza de código. Aquí está la clase de página para el código de sólo **BasicStyleCode** proyecto. El constructor de la **BasicStyleCodePage** clase utiliza sintaxis de objetos-inicialización para imitar la sintaxis XAML en la definición de la Estilo objeto y aplicándolo a tres botones:

```

clase pública BasicStyleCodePage : Pagina de contenido
{
    público BasicStyleCodePage ()
    {
        recursos = nuevo ResourceDictionary
        {
            {"ButtonStyle", nuevo Estilo ( tipo de ( Botón ))
            {
                setters =
                {
                    nuevo Setter
                    {
                        propiedad = Ver .HorizontalOptionsProperty,
                        valor = LayoutOptions .Centrar
                    },
                    nuevo Setter
                    {
                        propiedad = Ver .VerticalOptionsProperty,
                        valor = LayoutOptions .CenterAndExpand
                    },
                    nuevo Setter
                    {
                        propiedad = Botón .BorderWidthProperty,
                        Valor = 3
                    },
                    nuevo Setter
                    {
                        propiedad = Botón .TextColorProperty,
                        valor = Color .Rojo
                    },
                    nuevo Setter
                    {
                        propiedad = Botón .FontSizeProperty,
                        valor = Dispositivo .GetNamedSize ( NamedSize .Grande, tipo de ( Botón ))
                    },
                    nuevo Setter
                    {
                        Propiedad = VisualElement.BackgroundColorProperty,
                        Valor = Device.OnPlatform ( Color.Default,

```

```
        },
        nuevo Setter
        {
            Propiedad = Button.BorderColorProperty,
            Valor = Device.OnPlatform (Color.Default,
                                         Color blanco,
                                         De color negro)
        }
    }
}

};

content = nuevo StackLayout
{
    Los niños =
    {
        nuevo Botón
        {
            text = "Carpe Diem",
            Style = ( Estilo [Recursos] "ButtonStyle" )
        },
        nuevo Botón
        {
            text = "Sapere aude",
            Style = ( Estilo [Recursos] "ButtonStyle" )
        },
        nuevo Botón
        {
            text = "Faciendo discere",
            Style = ( Estilo [Recursos] "ButtonStyle" )
        }
    }
};
```

Es mucho más evidente en el código XAML que en que la Propiedad propiedad de la Setter es tipo BindableProperty.

Los primeros dos `Setter` objetos en este ejemplo se inicializan con el `BindableProperties` objetos llamado `View.HorizontalOptionsProperty` y `View.VerticalOptionsProperty`. Se podría utilizar `Button.HorizontalOptionsProperty` y `Button.VerticalOptionsProperty` en su lugar porque Botón hereda estas propiedades desde Ver. O bien, puede cambiar el nombre de la clase a cualquier otra clase que deriva de Ver.

Como de costumbre, el uso de una ResourceDictionary en el código parece inútil. Se podría eliminar el diccionario y simplemente asignar el Estilo objetos directamente a la Estilo propiedades de los botones. Sin embargo, incluso en el código, el Estilo es una manera conveniente de agrupar todos los valores de propiedades juntos en un paquete compacto.

herencia de estilos

los **Tipo de objetivo del Estilo** sirve para dos funciones diferentes: Una de estas funciones se describe en la siguiente sección sobre estilos implícitos. La otra función es para el beneficio del analizador XAML. El analizador XAML debe ser capaz de resolver los nombres de propiedad en el Setter objetos, y para ello se necesita un nombre de clase proporcionada por el **Tipo de objetivo**.

Todas las propiedades en el estilo deben ser definidos por o heredados por la clase especificada en el **Objetivo-Tipo** propiedad. El tipo del elemento visual en la que el Estilo está ajustado debe ser el mismo que el **Alquitrán-getType** o una clase derivada de la **Tipo de objetivo**.

Si necesita una Estilo sólo para propiedades definidas por Ver, se puede establecer el **Tipo de objetivo** a Ver y seguir utilizando el estilo en los botones o cualquier otra Ver derivado, como en este versión modificada de la **BasicStyle** programa:

```
< Página de contenido xmlns = "http://xamarin.com/schemas/2014/forms" 
    xmlns: x = "http://schemas.microsoft.com/winfx/2009/xaml" 
    x: Class = "BasicStyle.BasicStylePage" >

    < ContentPage.Resources >
        < ResourceDictionary >
            < Estilo x: Key = "ViewStyle" Tipo de objetivo = "Ver" >
                < Setter Propiedad = "HorizontalOptions" Valor = "Centrar" />
                < Setter Propiedad = "VerticalOptions" Valor = "CenterAndExpand" />
                < Setter Propiedad = "Color de fondo" Valor = "Rosado" />
            </ Estilo >
        </ ResourceDictionary >
    </ ContentPage.Resources >

    < StackLayout >
        < Botón Texto = "Cerpe Diem" 
            Estilo = "{} StaticResource ViewStyle" />

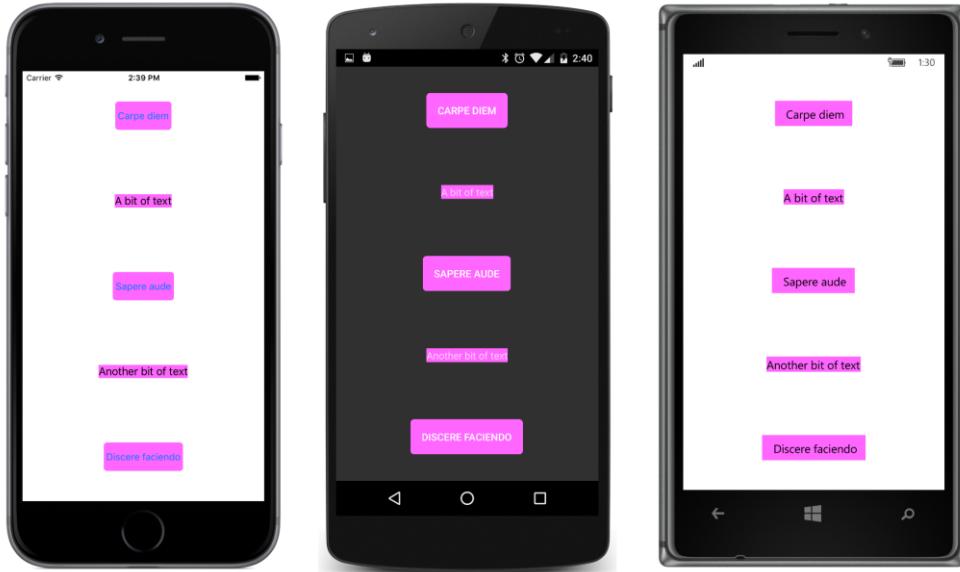
        < Etiqueta Texto = "Un fragmento de texto" 
            Estilo = "{} StaticResource ViewStyle" />

        < Botón Texto = "Sapere aude" 
            Estilo = "{} StaticResource ViewStyle" />

        < Etiqueta Texto = "Otro poco de texto" 
            Estilo = "{} StaticResource ViewStyle" />

        < Botón Texto = "faciendo discere" 
            Estilo = "{} StaticResource ViewStyle" />
    </ StackLayout >
</ Página de contenido >
```

Como se puede ver, el mismo estilo se aplica a toda la Botón y Etiqueta niños de la StackLayout:



Pero supongamos que ahora desea ampliar en este estilo, pero de forma diferente para Botón y Etiqueta. ¿Es eso posible?

Si lo es. Los estilos pueden derivar de otros estilos. Los Estilo clase incluye una propiedad denominada Residencia en de tipo Estilo. En el código, puede establecer esta Residencia en la propiedad directamente a otro Estilo objeto. En XAML se establece la Residencia en atribuir a una StaticResource extensión de marcado que hace referencia a una previamente creado Estilo. El nuevo Estilo puede incluir Setter Objetos para nuevas propiedades o utilizarlos para anular las propiedades en el anterior Estilo. Los Residencia en estilo debe apuntar a la misma clase o de una clase ancestro del nuevo estilo de Tipo de objetivo.

Aquí está el archivo XAML para un proyecto llamado **StyleInheritance**. La aplicación tiene una referencia a la **Xamarin.FormsBook.Toolkit** de montaje para dos propósitos: Se utiliza el HslColor extensión de marcado para demostrar que son extensiones de marcado ajustes de valores legítimos en Setter objetos y para demostrar que un estilo se puede definir para una clase personalizada, en este caso AltLabel.

Los ResourceDictionary contiene cuatro estilos: El primero tiene una clave de diccionario de "visualStyle". Los Estilo con la clave de diccionario de "baseStyle" deriva de "visualStyle". Los estilos con teclas de "LabelStyle" y "buttonStyle" se derivan de "baseStyle":

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    xmlns: kit de herramientas =
        " cl-espacio de nombres: Xamarin.FormsBook.Toolkit; montaje = Xamarin.FormsBook.Toolkit "
    x: Class = " StyleInheritance.StyleInheritancePage ">

< ContentPage.Resources >
    < ResourceDictionary >
        < Estilo x: Key = " estilo visual " Tipo de objetivo = " VisualElement " >
```

```

< Setter Propiedad = "Color de fondo "
  Valor = " {Toolkit: HslColor H = 0, S = 1, L = 0,8} " />
</ Estilo >

< Estilo x: Key = " baseStyle " Tipo de objetivo = " Ver "
  Residencia en = " {} StaticResource visualStyle " >
  < Setter Propiedad = " HorizontalOptions " Valor = " Centrar " />
  < Setter Propiedad = " VerticalOptions " Valor = " CenterAndExpand " />
</ Estilo >

< Estilo x: Key = " LabelStyle " Tipo de objetivo = " kit de herramientas: altLabel "
  Residencia en = " {} StaticResource baseStyle " >
  < Setter Propiedad = " Color de texto " Valor = " Negro " />
  < Setter Propiedad = " PointSize " Valor = " 12 " />
</ Estilo >

< Estilo x: Key = " buttonStyle " Tipo de objetivo = " Botón "
  Residencia en = " {} StaticResource baseStyle " >
  < Setter Propiedad = " Color de texto " Valor = " Azul " />
  < Setter Propiedad = " Tamaño de fuente " Valor = " Grande " />
  < Setter Propiedad = " Color del borde " Valor = " Azul " />
  < Setter Propiedad = " Ancho del borde " Valor = " 2 " />
</ Estilo >
</ ResourceDictionary >
</ ContentPage.Resources >

< ContentPage.Style >
  < StaticResourceExtension Llave = " estilo visual " />
</ ContentPage.Style >

< StackLayout >
  < Botón Texto = " Carpe Diem "
    Estilo = " {} StaticResource buttonStyle " />

  < kit de herramientas: altLabel Texto = " Un fragmento de texto "
    Estilo = " {} StaticResource LabelStyle " />

  < Botón Texto = " Sapere aude "
    Estilo = " {} StaticResource buttonStyle " />

  < kit de herramientas: altLabel Texto = " Otro poco de texto "
    Estilo = " {} StaticResource LabelStyle " />

  < Botón Texto = " faciendo discere "
    Estilo = " {} StaticResource buttonStyle " />
</ StackLayout >
</ Pagina de contenido >

```

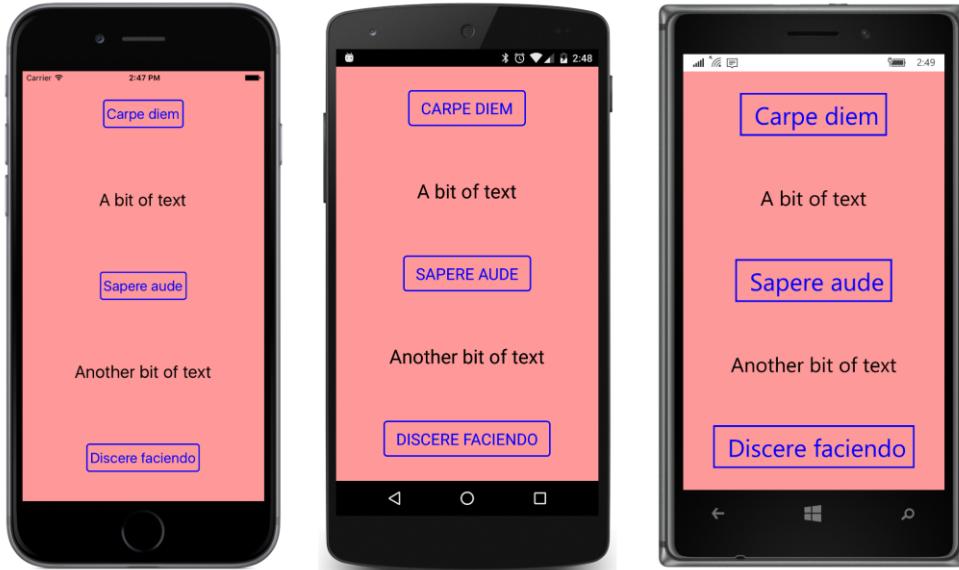
Inmediatamente después de la sección de recursos es un poco marcado que establece el **Estilo** propiedad de la propia página de la "visualStyle" Estilo:

```

< ContentPage.Style >
  < StaticResourceExtension Llave = " estilo visual " />
</ ContentPage.Style >

```

Porque Página deriva de `VisualElement` pero no `Ver`, este es el único estilo en el diccionario de recursos que se pueden aplicar a la página. Sin embargo, el estilo no se puede aplicar a la página hasta después de la `Ruentes sección`, así que usar la forma de elemento `StaticResource` es una buena solución a este problema. todo el fondo de la página es de color basado en este estilo, y el estilo también es heredado por todos los otros estilos:



Si el Estilo Para el `AltLabel` Sólo se incluye `Setter` objetos para propiedades definidas por Etiqueta, el Tipo de objetivo podría ser Etiqueta en lugar de `AltLabel`. Pero el Estilo tiene un `Setter` Para el `PointSize` propiedad. Esa propiedad se define por `AltLabel`, entonces el Tipo de objetivo debe ser kit de herramientas: `AltLabel`.

UN `Setter` se puede definir para el `PointSize` propiedad porque `PointSize` está respaldado por una propiedad enlazable. Si cambia la accesibilidad de la `BindableProperty` oponerse en `AltLabel` de público a privado, la propiedad seguirá funcionando para muchos usos de rutina de `AltLabel`, pero ahora `PointSize` no se puede establecer en un estilo `Setter`. El analizador de XAML se quejará de que no puede encontrar `PointSizeProperty`, que es la propiedad enlazable que respalda la `PointSize` propiedad.

Has descubierto cómo en el capítulo 10 `StaticResource` funciona: Cuando el analizador XAML se encuentra con una `StaticResource` extensión de marcado, se busca en el árbol visual para una llave diccionario. Este proceso tiene implicaciones para los estilos. Se puede definir un estilo en una recursos sección y luego sustituirlo por otro estilo con la misma clave en un diccionario diferente recursos la sección inferior en el árbol visual. Cuando se establece el Residencia en propiedad a un `StaticResource` extensión de marcado, el estilo que está derivado de se debe definir en el mismo recursos sección (como se demuestra en el

`StyleInheritance` programa) o una recursos la sección superior en el árbol visual.

Esto significa que se puede estructurar sus estilos en XAML de dos formas jerárquicas: Puede utilizar `Residencia en` para derivar estilos de otros estilos, y se puede definir estilos en los diferentes niveles de la calidad visual

árbol que se derivan de estilos más alto en el árbol visual o reemplazar por completo.

Para aplicaciones más grandes con varias páginas y un montón de marcas, la recomendación para los estilos que definen es muy simple-definir sus estilos lo más cerca posible a los elementos que utilizan esos estilos.

Al cumplir con esta recomendación ayuda a mantener el programa y se vuelve particularmente importante cuando se trabaja con *estilos implícitos*.

estilos implícitos

Cada entrada en una `ResourceDictionary` requiere una clave de diccionario. Este es un hecho indiscutible. Si intenta pasar una nula clave para la `Añadir` método de una `ResourceDictionary` objeto, te plantean una `ArgumentNullException`.

Sin embargo, hay un caso especial en el que no se requiere un programador para suministrar esta clave de diccionario. Una clave de diccionario en su lugar se genera automáticamente.

Este caso especial es para una Estilo objeto añadió a una `ResourceDictionary` sin una `x: Key` ajuste. Los `ResourceDictionary` genera una clave basada en el Tipo de objetivo, que siempre se requiere. (Un poco de exploración revelará que esta llave especial diccionario es el nombre completo asociado con el Tipo de objetivo del Estilo. Para Tipo de objetivo de Botón, por ejemplo, la clave de diccionario es "Xamarin.Forms.Button". Pero usted no necesita saber eso.)

También puede agregar una Estilo a una `ResourceDictionary` sin una llave en el diccionario de código: una sobrecarga de la `Añadir` método acepta un argumento de tipo Estilo pero no requiere ninguna otra cosa.

UN Estilo objeto en una `ResourceDictionary` que tiene una de estas claves generadas se conoce como una *estilo implícita*, y la clave de diccionario generado es muy especial. No se puede hacer referencia a esta clave utilizando directamente `StaticResource`. Sin embargo, si un elemento dentro del alcance de la `ResourceDictionary` tiene el mismo tipo que la clave de diccionario, y si ese elemento no tiene su Estilo establece explícitamente la propiedad a otro Estilo objeto, entonces este estilo implícita se aplica automáticamente.

El siguiente XAML de la `ImplicitStyle` proyecto demuestra esto. Es el mismo que el `BasicStyle` archivo XAML excepto que el Estilo no tiene `x: Key` entorno y la Estilo propiedades de los botones no se establecen con `StaticResource`:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " ImplicitStyle.ImplicitStylePage " >

    < ContentPage.Resources >
        < ResourceDictionary >
            < Estilo Tipo de objetivo = " Botón " >
                < Setter Propiedad = " HorizontalOptions " Valor = " Centrar " />
                < Setter Propiedad = " VerticalOptions " Valor = " CenterAndExpand " />
                < Setter Propiedad = " Ancho del borde " Valor = " 3 " />
                < Setter Propiedad = " Color de texto " Valor = " rojo " />
```

```
<Setter Propiedad = "Tamaño de fuente" Valor = "Grande" />
<Setter Propiedad = "Color de fondo" >
    <Setter.Value>
        <OnPlatform x:TypeArguments = "Color" 
            Androide = "#404040" />
    </Setter.Value>
</Setter>

<Setter Propiedad = "Color del borde" >
    <Setter.Value>
        <OnPlatform x:TypeArguments = "Color" 
            Androide = "Blanco" 
            WinPhone = "Negro" />
    </Setter.Value>
</Setter>
</Estilo>
</ResourceDictionary>
</ContentPage.Resources>

<StackLayout >
    <Botón Texto = "Carpe Diem" />
    <Botón Texto = "Sapere audē" />
    <Botón Texto = "faciendo discere" />
</StackLayout>
</Pagina de contenido>
```

A pesar de la ausencia de una conexión explícita entre los botones y el estilo, el estilo se aplica definitivamente:



Un estilo implícita se aplica sólo cuando la clase del elemento coincide con el Tipo de objetivo del Estilo exactamente. Si se incluye un elemento que se deriva de Botón en el StackLayout, no tendría la Estilo aplicado.

Puede utilizar la configuración de propiedades locales para anular las propiedades establecidas a través del estilo implícita, tal como se puede anular la configuración de propiedad en un conjunto con estilo StaticResource.

Usted encontrará estilos implícitos a ser muy potente y extremadamente útil. Siempre que tenga varias vistas del mismo tipo y determina que usted quiere que todos ellos tienen un valor de la propiedad idéntica o dos, es muy fácil de definir rápidamente un estilo implícita. Usted no tiene que tocar los elementos mismos.

Sin embargo, un gran poder conlleva al menos *algunos* responsabilidad del programador. Debido a que ningún estilo se hace referencia en los propios elementos, puede ser confuso al examinar simplemente el XAML para determinar si algunos elementos son de estilo o no. A veces la apariencia de una página indica que un estilo implícita se aplica a algunos elementos, pero no es bastante obvio donde se define el estilo implícita. Si a continuación desea cambiar ese estilo implícita, usted tiene que buscar manualmente para que el árbol visual.

Por esta razón, se deben definir los estilos implícitos *Tan cerca como sea posible* a los elementos que se aplican a. Si los puntos de vista para conseguir el estilo implícita se encuentran en una determinada StackLayout, a continuación, definir el estilo implícito en el recursos sección en ese StackLayout. Un par de comentarios puede ayudar a evitar la confusión también.

Curiosamente, estilos implícitos tienen una restricción incorporada que podría persuadir a mantenerlos cerca de los elementos que se aplican a. Aquí está la restricción: Se puede derivar un estilo implícita de una Estilo con una clave de diccionario explícita, pero no se puede ir al revés. No se puede utilizar Residencia en para hacer referencia a un estilo implícita.

Si se define una cadena de estilos que utilizan Residencia en para derivar el uno del otro, el estilo implícita (si existe) es siempre al final de la cadena. No hay más derivaciones son posibles.

Esto implica que se puede estructurar sus estilos con tres tipos de jerarquías:

- A partir de estilos definidos en el Solicitud y Página a los estilos definidos en los diseños más baja en el árbol visual.
- A partir de estilos definidos para las clases de base, tales como VisualElement y Ver a los estilos definidos para las clases específicas.
- De estilos con explícitas claves de diccionario a estilos implícitos.

Esto se demuestra en el **StyleHierarchy** proyecto, que utiliza un conjunto similar (pero algo simplificada) de estilos como se vio anteriormente en el **StyleInheritance** proyecto. Sin embargo, estos estilos están repartidos en tres recursos secciones.

Usando una técnica que viste en el **ResourceTrees** programa en el capítulo 10, el **StyleHierarchy** proyecto se le dio una XAML-basa Aplicación clase. La clase tiene una App.xaml ResourceDictionary que contiene un estilo con colocador sólo una propiedad:

```
< Solicitud xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns:x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " StyleHierarchy.App " >

    < Application.Resources >
        < ResourceDictionary >
            < Estilo x: Key = " estilo visual " Tipo de objetivo = " VisualElement " >
                < Setter Propiedad = " Color de fondo " Valor = " Rosado " />
            </ Estilo >
        </ ResourceDictionary >
    </ Application.Resources >
</ Solicitud >
```

En una aplicación de varias páginas, este estilo se usa en toda la aplicación.

El archivo de código subyacente para el Aplicación llamadas de clase InitializeComponent para procesar el archivo XAML y establece el Página principal propiedad:

```
público clase parcial Aplicación : Solicitud
{
    público App ()
    {
        InitializeComponent ();
        MainPage = nuevo StyleHierarchyPage ();
    }
    ...
}
```

El archivo XAML para la clase de página define una Estilo para toda la página que se deriva del estilo en el Aplicación clase y también dos estilos implícitas que se derivan de la Estilo para la página. Observe que el Estilo propiedad de la página se establece en el Estilo se define en la Aplicación clase:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns:x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " StyleHierarchy.StyleHierarchyPage "
    Estilo = " { } StaticResource visualStyle " >

    < ContentPage.Resources >
        < ResourceDictionary >
            < Estilo x: Key = " baseStyle " Tipo de objetivo = " Ver "
                Residencia en = " { } StaticResource visualStyle " >
                < Setter Propiedad = " HorizontalOptions " Valor = " Centrar " />
                < Setter Propiedad = " VerticalOptions " Valor = " CenterAndExpand " />
            </ Estilo >
        </ ResourceDictionary >
    </ ContentPage.Resources >

    < StackLayout >
        < StackLayout.Resources >
```

```
< ResourceDictionary >
    < Estilo Tipo de objetivo = " Etiqueta " 
        Residencia en = " {} StaticResource baseStyle " >
        < Setter Propiedad = " Color de texto " Valor = " Negro " />
        < Setter Propiedad = " Tamaño de fuente " Valor = " Grande " />
    </ Estilo >

    < Estilo Tipo de objetivo = " Botón " 
        Residencia en = " {} StaticResource baseStyle " >
        < Setter Propiedad = " Color de texto " Valor = " Azul " />
        < Setter Propiedad = " Tamaño de fuente " Valor = " Grande " />
        < Setter Propiedad = " Color del borde " Valor = " Azul " />
        < Setter Propiedad = " Ancho del borde " Valor = " 2 " />
    </ Estilo >
</ ResourceDictionary >
</ StackLayout.Resources >

< Botón Texto = " Carpe Diem " />

< Etiqueta Texto = " Un fragmento de texto " />

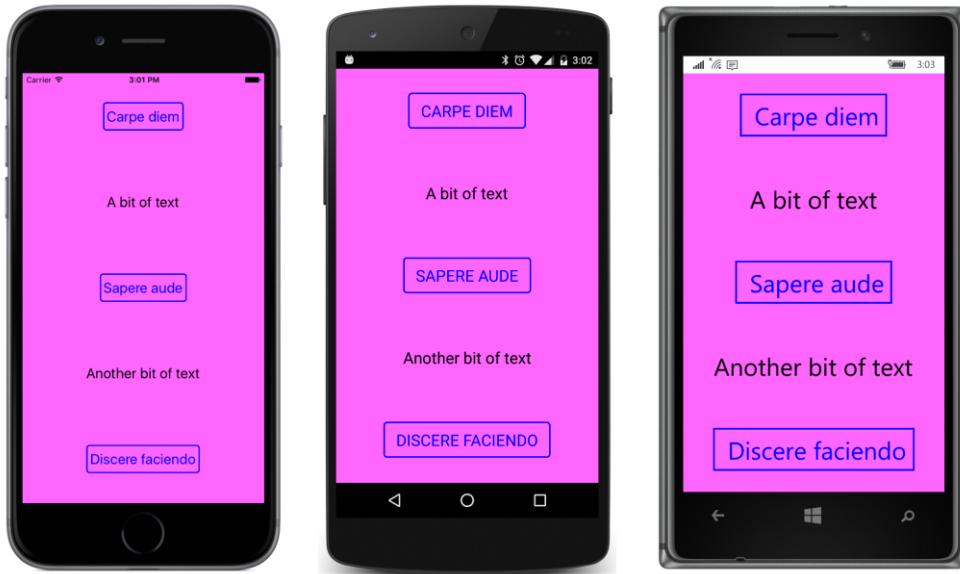
< Botón Texto = " Sapere aude " />

< Etiqueta Texto = " Otro poco de texto " />

< Botón Texto = " faciendo discere " />
</ StackLayout >
</ Pagina de contenido >
```

Los estilos implícitos se definen como cerca de los elementos diana como sea posible.

Aquí está el resultado:



El incentivo para separar Estilo objetos en distintos diccionarios no hace mucho sentido para muy pequeños programas como éste, pero para los programas de mayor tamaño, se hace tan importante tener una jerarquía estructurada de las definiciones de estilo, ya que es tener una jerarquía estructurada de las definiciones de clase.

A veces, usted tendrá una Estilo con un diccionario explícita clave (por ejemplo, "myButtonStyle"), pero que querrá ese mismo estilo que sea implícita también. Basta con definir un estilo basado en esa tecla sin llave o definidores de su propia:

```
<Estilo Tipo de objetivo = "Botón"
      Residencia en = "{} StaticResource myButtonStyle" />
```

Eso es un estilo implícita de que es idéntica a myButtonStyle.

estilos dinámicos

UN Estilo es generalmente un objeto estático que se crea y se inicializa en XAML o código y luego se mantiene sin cambios durante la duración de la aplicación. Los Estilo clase no se deriva de BindableObject y no responde a los cambios internos en sus propiedades. Por ejemplo, si asigna una Estilo a un elemento y, a continuación modificar una de las Setter objetos, dándole un nuevo valor, el nuevo valor no se mostrarán en el elemento. Del mismo modo, el elemento de destino no cambiará si se agrega una Setter o eliminar una Setter desde el setters colección. Para estos nuevos emisores de propiedad surtan efecto, es necesario utilizar código para separar el estilo del elemento fijando el Estilo propiedad a nulo y luego vuelva a colocar el estilo al elemento.

Sin embargo, su aplicación puede responder a cambios de estilo de forma dinámica en tiempo de ejecución mediante el uso de **DynamicResource**. Usted recordará que **DynamicResource** es parecido a **StaticResource** ya que utiliza una clave de diccionario a buscar un objeto o un valor de un diccionario de recursos. La diferencia es que Estático-Recurso es una sola vez, mientras que el diccionario de búsqueda **DynamicResource** mantiene un enlace a la clave de diccionario real. Si la entrada del diccionario asociado con esa clave se sustituye por un nuevo objeto, que el cambio se propaga al elemento.

Esta facilidad permite que una aplicación para implementar una característica a veces llamado *estilos dinámicos*. Por ejemplo, podría incluir una facilidad en su programa para los temas estilísticos (que implican fuentes y colores, tal vez), y es posible que estos temas seleccionables por el usuario. La aplicación puede cambiar entre estos temas, ya que se implementan con estilos.

No hay nada en un mismo estilo que indica un estilo dinámico. Un estilo dinámico se convierte únicamente por ser referenciado utilizando **DynamicResource** más bien que **StaticResource**.

los **DynamicStyles** proyecto demuestra la mecánica de este proceso. Aquí está el archivo XAML para el **DynamicStylesPage** clase:

```
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " DynamicStyles.DynamicStylesPage " >

    < ContentPage.Padding >
        < OnPlatform x: TypeArguments = " Espesor "
            iOS = " 0, 20, 0, 0 "
            Androide = " 0 "
            WinPhone = " 0 " />
    </ ContentPage.Padding >

    < ContentPage.Resources >
        < ResourceDictionary >
            < Estilo x: Key = " baseButtonStyle " Tipo de objetivo = " Botón " >
                < Setter Propiedad = " Tamaño de fuente " Valor = " Grande " />
            </ Estilo >

            < Estilo x: Key = " buttonStyle1 " Tipo de objetivo = " Botón "
                Residencia en = " () StaticResource baseButtonStyle " >
                < Setter Propiedad = " HorizontalOptions " Valor = " Centrar " />
                < Setter Propiedad = " VerticalOptions " Valor = " CenterAndExpand " />
                < Setter Propiedad = " Color de texto " Valor = " rojo " />
            </ Estilo >

            < Estilo x: Key = " buttonStyle2 " Tipo de objetivo = " Botón "
                Residencia en = " () StaticResource baseButtonStyle " >
                < Setter Propiedad = " HorizontalOptions " Valor = " comienzo " />
                < Setter Propiedad = " VerticalOptions " Valor = " EndAndExpand " />
                < Setter Propiedad = " Color de texto " Valor = " Verde " />
                < Setter Propiedad = " FontAttributes " Valor = " Itálico " />
            </ Estilo >

            < Estilo x: Key = " buttonStyle3 " Tipo de objetivo = " Botón "

```

```
Residencia en = " {StaticResource baseButtonStyle} "
<Setter Propiedad = "HorizontalOptions" Valor = "Fin" />
<Setter Propiedad = "VerticalOptions" Valor = "StartAndExpand" />
<Setter Propiedad = "Color de texto" Valor = "Azul" />
<Setter Propiedad = "FontAttributes" Valor = "Negrita" />

</Estilo>
</ResourceDictionary>
</ContentPage.Resources>

<StackLayout>
<Botón Texto = "Cambiar al estilo # 1"
Estilo = " {DynamicResource buttonStyle} "
hecho clic = "OnButton1Clicked" />

<Botón Texto = "Cambiar al estilo # 2"
Estilo = " {DynamicResource buttonStyle} "
hecho clic = "OnButton2Clicked" />

<Botón Texto = "Cambiar al estilo # 3"
Estilo = " {DynamicResource buttonStyle} "
hecho clic = "OnButton3Clicked" />

<Botón Texto = "Reiniciar"
Estilo = " {DynamicResource buttonStyle} "
hecho clic = "OnResetButtonClicked" />
</StackLayout>
</Pagina de contenido>
```

los recursos sección define cuatro estilos: un estilo sencillo con la tecla “baseButtonStyle”, y luego tres estilos que se derivan de ese estilo con las teclas “buttonStyle1”, “buttonStyle2”, y “buttonStyle3”.

Sin embargo, los cuatro Botón elementos hacia la parte inferior del archivo XAML todo el uso DynamicResource para hacer referencia a un estilo más simple con la tecla “buttonStyle”. Dónde está el Estilo con esa clave? No existe. Sin embargo, debido a que el botón de cuatro Estilo propiedades se establecen con DynamicResource, la clave de diccionario que falta no es un problema. Sin excepción se lanza. Pero no Estilo se aplican, lo que significa que los botones tienen un aspecto por defecto:



Cada uno de los cuatro Botón elementos tiene una hecho clic controlador asociado, y en el archivo de código subyacente, los tres primeros manipuladores establece una entrada de diccionario con la tecla “buttonStyle” a uno de los tres estilos numerados ya definidos en el diccionario:

```
pública clase parcial DynamicStylesPage : Página de contenido
{
    pública DynamicStylesPage ()
    {
        InitializeComponent ();
    }

    vacío OnButton1Clicked ( objeto remitente, EventArgs args )
    {
        recursos [ "ButtonStyle" ] = Recursos [ "ButtonStyle1" ];
    }

    vacío OnButton2Clicked ( objeto remitente, EventArgs args )
    {
        recursos [ "ButtonStyle" ] = Recursos [ "ButtonStyle2" ];
    }

    vacío OnButton3Clicked ( objeto remitente, EventArgs args )
    {
        recursos [ "ButtonStyle" ] = Recursos [ "ButtonStyle3" ];
    }

    vacío OnResetButtonClicked ( objeto remitente, EventArgs args )
    {
        recursos [ "ButtonStyle" ] = nulo ;
    }
}
```

Cuando se pulsa uno de los tres primeros botones, los cuatro botones reciben el estilo seleccionado. Aquí está el programa que se ejecuta en las tres plataformas que muestran los resultados (de izquierda a derecha) cuando los botones 1, 2 y 3 se presionan:



Al pulsar el cuarto botón vuelve todo a las condiciones iniciales estableciendo el valor asociado a la clave “buttonStyle” a nulo. (También puede considerar llamar `retirar o Claro sobre el ResourceDictionary` objeto de eliminar la clave del todo, pero que no funciona en la versión de Xamarin.Forms utilizadas para este capítulo).

Supongamos que se desea derivar otra Estilo desde el Estilo con la tecla “buttonStyle”. ¿Cómo se hace esto en XAML, teniendo en cuenta que el “buttonStyle” entrada de diccionario no existe hasta que se pulsa uno de los tres primeros botones?

No se puede hacer así:

```
<! - Esto no va a funcionar! ->
<Estilo x:Key = "newButtonStyle" Tipo de objetivo = "Botón"
      Residencia en = "{} StaticResource buttonStyle" >
...
</Estilo >
```

`StaticResource` lanzará una excepción si no existe la tecla “buttonStyle”, e incluso si no existe la clave, el uso de `StaticResource` no permitirá cambios en la entrada del diccionario que se reflejan en este nuevo estilo.

Sin embargo, cambiar `StaticResource` a `DynamicResource` no va a funcionar, ya sea:

```
<! - Esto no funcionará, ya sea! ->
<Estilo x:Key = "newButtonStyle" Tipo de objetivo = "Botón"
```

```

Residencia en = " {} DynamicResource buttonStyle " >
...
</ Estilo >
```

DynamicResource Sólo funciona con propiedades respaldados por propiedades enlazables, y ese no es el caso aquí. Estilo no se deriva de **bindableObject** por lo que no puede soportar propiedades enlazables.

En lugar, Estilo define una propiedad específicamente con el fin de heredar estilos dinámicos. La propiedad es **BaseResourceKey**, que está destinado a ser fijado directamente a una clave de diccionario que aún no podría existir o cuyo valor podría cambiar de forma dinámica, que es el caso con la tecla “buttonStyle”:

```

<! - ||Esto funcional! >
< Estilo x: Key = " newButtonStyle " Tipo de objetivo = " Botón "
  BaseResourceKey = " buttonStyle " >
...
</ Estilo >
```

El uso de **BaseResourceKey** se demuestra por la **DynamicStylesInheritance** proyecto, que es muy similar a la **DynamicStyles** proyecto. De hecho, el procesamiento de código subyacente es idéntica. Hacia el final de la recursos sección, un nuevo Estilo se define con una clave de “newButtonStyle” que utiliza

BaseResourceKey para hacer referencia a la entrada “buttonStyle” y añadir un par de propiedades, incluyendo uno que utiliza **OnPlatform**:

```

< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
  xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
  x: Class = " DynamicStylesInheritance.DynamicStylesInheritancePage " >
< ContentPage.Padding >
  < OnPlatform x: TypeArguments = " Espesor "
    iOS = " 0, 20, 0, 0 "
    Androide = " 0 "
    WinPhone = " 0 " />
</ ContentPage.Padding >

< ContentPage.Resources >
  < ResourceDictionary >
    < Estilo x: Key = " baseButtonStyle " Tipo de objetivo = " Botón " >
      < Setter Propiedad = " Tamaño de fuente " Valor = " Grande " />
    </ Estilo >

    < Estilo x: Key = " buttonStyle1 " Tipo de objetivo = " Botón "
      Residencia en = " {} StaticResource baseButtonStyle " >
      < Setter Propiedad = " HorizontalOptions " Valor = " Centro " />
      < Setter Propiedad = " VerticalOptions " Valor = " CenterAndExpand " />
      < Setter Propiedad = " Color de texto " Valor = " rojo " />
    </ Estilo >

    < Estilo x: Key = " buttonStyle2 " Tipo de objetivo = " Botón "
      Residencia en = " {} StaticResource baseButtonStyle " >
      < Setter Propiedad = " HorizontalOptions " Valor = " comienzo " />
      < Setter Propiedad = " VerticalOptions " Valor = " EndAndExpand " />
      < Setter Propiedad = " Color de texto " Valor = " Verde " />
      < Setter Propiedad = " FontAttributes " Valor = " Itálico " />
    </ Estilo >
```

```

</ Estilo >

< Estilo x: Key = "buttonStyle3" Tipo de objetivo = "Botón" 
    Residencia en = " {} StaticResource baseButtonStyle "
    < Setter Propiedad = " HorizontalOptions " Valor = " Fin " />
    < Setter Propiedad = " VerticalOptions " Valor = " StartAndExpand " />
    < Setter Propiedad = " Color de texto " Valor = " Azul " />
    < Setter Propiedad = " FontAttributes " Valor = " Negrita " />
</ Estilo >

<!-- Nueva definición de estilo. -->
< Estilo x: Key = "newButtonStyle" Tipo de objetivo = "Botón" 
    BaseResourceKey = " buttonStyle "
    < Setter Propiedad = " Color de fondo " >
        < Setter.Value >
            < OnPlatform x: TypeArguments = " Color "
                iOS = "# C0C0C0 "
                Androide = "# 404040 "
                WinPhone = " gris " />
            </ Setter.Value >
        </ Setter >
        < Setter Propiedad = " Color del borde " Valor = " rojo " />
        < Setter Propiedad = " Ancho del borde " Valor = " 3 " />
    </ Estilo >
</ ResourceDictionary >
</ ContentPage.Resources >

< StackLayout >
    < Botón Texto = " Cambiar al estilo # 1 "
        Estilo = " {} StaticResource newButtonStyle "
        hecho clic = " OnButton1Clicked " />

    < Botón Texto = " Cambiar al estilo # 2 "
        Estilo = " {} StaticResource newButtonStyle "
        hecho clic = " OnButton2Clicked " />

    < Botón Texto = " Cambiar al estilo # 3 "
        Estilo = " {} StaticResource newButtonStyle "
        hecho clic = " OnButton3Clicked " />

    < Botón Texto = " Reiniciar "
        Estilo = " {} DynamicResource buttonStyle "
        hecho clic = " OnResetButtonClicked " />
</ StackLayout >
</ Página de contenido >

```

Tenga en cuenta que los tres primeros Botón Referencia a elementos de la "newButtonStyle" con entrada de diccionario StaticResource. DynamicResource No se necesita aquí porque el Estilo objeto asociado al "newButtonStyle" no es en sí cambiará a excepción de la Estilo que deriva de. los Estilo con la tecla "newButtonStyle" mantiene un vínculo con "buttonStyle" e internamente altera cuando cambia ese mismo estilo subyacentes. Cuando el programa comienza a funcionar, solamente las propiedades definidas en el "newButtonStyle" se aplican a esos tres botones:



los **Reiniciar Botón** continúa haciendo referencia a la entrada "buttonStyle".

Como en el **DynamicStyles** programa, el archivo de código subyacente establece que la entrada de diccionario al hacer clic en uno de los tres primeros botones, por lo que todos los botones de recoger las propiedades "ButtonStyle" también. Aquí están los resultados de (de izquierda a derecha) clics de botones 3, 2 y 1:



estilos de dispositivos

Xamarin.Forms incluye seis estilos dinámicos incorporados. Estos son conocidos como *estilos de dispositivos*, y son miembros de una clase anidada de Dispositivo llamado **Estilos**. Esta clase define 12 estéticos y solo lectura campos que ayudan a hacer referencia a estos seis estilos en el código:

- **Tipo de cuerpo de tipo Estilo.**
- **BodyStyleKey de tipo cuerda e igual a "estilo de carrocería."**
- **TitleStyle de tipo Estilo.**
- **TitleStyleKey de tipo cuerda e igual a "TitleStyle."**
- **SubtitleStyle de tipo Estilo.**
- **SubtitleStyleKey de tipo cuerda e igual a "SubtitleStyle."**
- **CaptionStyle de tipo Estilo.**
- **CaptionStyleKey de tipo cuerda e igual a "CaptionStyle."**
- **ListItemTextStyle de tipo Estilo.**
- **ListItemTextStyleKey de tipo cuerda e igual a "ListItemTextStyle."**
- **ListItemDetailTextStyle de tipo Estilo.**
- **ListItemDetailTextStyleKey de tipo cuerda e igual a "ListDetailTextStyle."**

Los seis estilos tienen una **Tipo de objetivo de Etiqueta** y se almacenan en un diccionario, pero no un diccionario que los programas de aplicación pueden acceder directamente.

En el código, utilice los campos en esta lista para acceder a los estilos de dispositivos. Por ejemplo, se puede establecer el **Device.Styles.BodyStyle** oponerse directamente a la **Estilo** propiedad de una Etiqueta para el texto que podrían ser apropiados para el cuerpo de un párrafo. Si define un estilo en el código que se deriva de uno de estos estilos dispositivo, establezca el **BaseResourceKey** a **Device.Styles.BodyStyleKey** o simplemente "Carrocería" si no tienes miedo de falta de ortografía ella.

En XAML, usted sólo tiene que utilizar la clave de texto "Carrocería" con **DynamicResource** para el establecimiento de este estilo a la **Estilo** propiedad de una Etiqueta o para establecer **BaseResourceKey** cuando se deriva de un estilo **Device.Styles.BodyStyle**.

los **DeviceStylesList** programa demuestra cómo acceder a estos estilos y para definir un nuevo estilo que hereda de **SubtitleStyle** -tanto en XAML y en el código. Aquí está el archivo XAML:

```
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns:x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " DeviceStylesList.DeviceStylesListPage ">
```

```
< ContentPage.Padding >
    < OnPlatform x: TypeArguments = " Espesor "
        iOS = " 10, 20, 10, 0 "
        Androide = " 10, 0 "
        WinPhone = " 10, 0 " />
</ ContentPage.Padding >

< ContentPage.Resources >
    < ResourceDictionary >
        < Estilo x: Key = " newSubtitleStyle " Tipo de objetivo = " Etiqueta "
            BaseResourceKey = " SubtitleStyle " >
            < Setter Propiedad = " Color de texto " Valor = " Acento " />
            < Setter Propiedad = " FontAttributes " Valor = " Itálico " />
        </ Estilo >
    </ ResourceDictionary >
</ ContentPage.Resources >

< ScrollView >
    < StackLayout Espaciado = " 20 " >

        <! - estilos de dispositivos de conjunto con DynamicResource ->
        < StackLayout >
            < StackLayout HorizontalOptions = " comienzo " >
                < Etiqueta Texto = " estilos de dispositivos de conjunto con DynamicResource " />
                < BoxView Color = " Acento " HeightRequest = " 3 " />
            </ StackLayout >

            < Etiqueta Texto = " Sin estilo en absoluto " />

            < Etiqueta Texto = " Tipo de cuerpo "
                Estilo = " {} DynamicResource bodystyle " />

            < Etiqueta Texto = " título Estilo "
                Estilo = " {} DynamicResource TitleStyle " />

            < Etiqueta Texto = " Tipo de subtítulos "
                Estilo = " {} DynamicResource SubtitleStyle " />

        <! - Usos estilo derivado de estilo dispositivo. ->
        < Etiqueta Texto = " Nuevo Tipo de subtítulos "
            Estilo = " {} StaticResource newSubtitleStyle " />

        < Etiqueta Texto = " Estilo del subtítulo "
            Estilo = " {} DynamicResource CaptionStyle " />

        < Etiqueta Texto = " Lista de elementos de estilo de texto "
            Estilo = " {} DynamicResource ListItemTextStyle " />

        < Etiqueta Texto = " Lista de elementos Detalle estilo de texto "
            Estilo = " {} DynamicResource ListItemDetailTextStyle " />
    </ StackLayout >

    <! - estilos dispositivo regulado de código ->
    < StackLayout x: Nombre = " codeLabelStack " >
```

```

< StackLayout HorizontalOptions = "comienzo" >
    < Etiqueta Texto = "estilos de dispositivos establecidos en el código: " />
    < BoxView Color = "Acento" HeightRequest = "3" />
</ StackLayout >
</ StackLayout >
</ ScrollView >
<Pagina de contenido>

```

los StackLayout contiene dos Etiqueta y BoxView combinaciones (uno en la parte superior y uno en la parte inferior) para visualizar los encabezados subrayados. Después de la primera de estas cabeceras, Etiqueta Referencia de los elementos de los estilos de dispositivo con DynamicResource. El nuevo estilo de los subtítulos se define en el recursos Inglés para la página.

El archivo de código subyacente accede a los estilos de dispositivos mediante el uso de las propiedades en el DeviceStyles clase y crea un nuevo estilo que deriva de SubtitleStyle:

```

pública clase parcial DeviceStylesListPage : Pagina de contenido
{
    pública DeviceStylesListPage ()
    {
        InitializeComponent ();

        var styleItems = nuevo []
        {
            nuevo (Style = ( Estilo ) nulo , Name = "No hay un estilo en absoluto" ),
            nuevo (Style = Dispositivo . estilos . BodyStyle, name = "Tipo de cuerpo" ),
            nuevo (Style = Dispositivo . estilos . TitleStyle, name = "Estilo de título" ),
            nuevo (Style = Dispositivo . estilos . SubtitleStyle, name = "Tipo de subtítulos" ),

            // estilo derivado
            nuevo (Style = nuevo Estilo ( tipo de ( Etiqueta ) )
            {
                BaseResourceKey = Dispositivo . estilos . SubtitleStyleKey,
                setters =
                {
                    nuevo Setter
                    {
                        propiedad = Etiqueta .TextColorProperty,
                        valor = Color .Acento
                    },
                    nuevo Setter
                    {
                        propiedad = Etiqueta .FontAttributesProperty,
                        valor = FontAttributes .Itálico
                    }
                }
            }, Name = "Nuevo Tipo de subtítulos" ),

            nuevo (Style = Dispositivo . estilos . CaptionStyle, name = "Estilo de subtítulos" ),
            nuevo (Style = Dispositivo . estilos . ListItemTextStyle, name = "Lista de elementos de estilo de texto" ),
            nuevo (Style = Dispositivo . estilos . ListItemDetailTextStyle,
                   name = "Lista de detalles de artículos de estilo de texto" ),
        }
    }
}

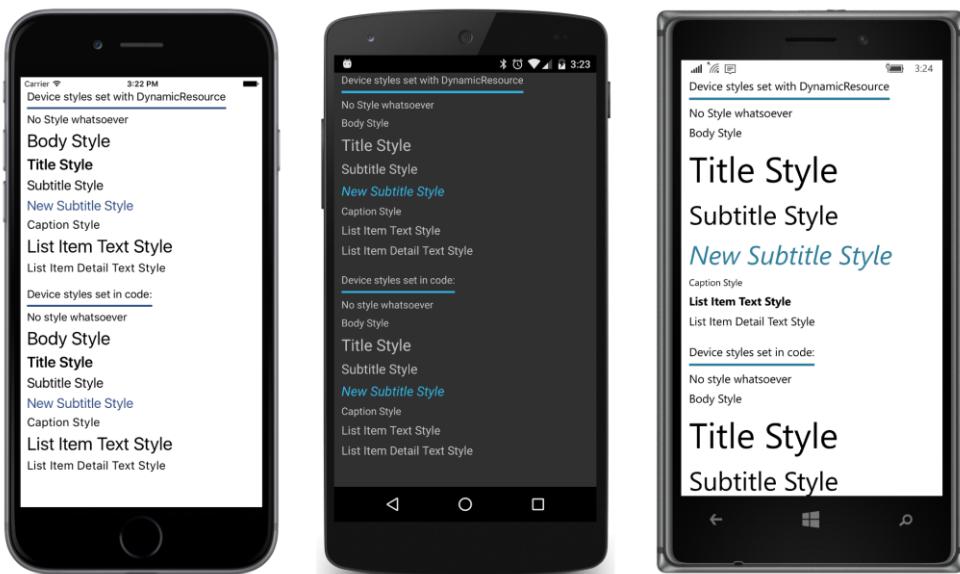
```

```
};

para cada ( var styleItem en styleItems)
{
    codeLabelStack.Children.Add ( nuevo Etiqueta
    {
        Text = styleItem.name,
        Style = styleItem.style
    });
}

}
}
```

El código XAML y el resultado de estilos idénticos, por supuesto, pero cada plataforma implementa estos estilos de dispositivos de una manera diferente:



La naturaleza dinámica de estos estilos se demuestra fácilmente en iOS: Mientras que el **DeviceStyles** programa se está ejecutando, toque el **Casa** botón y ejecución **Ajustes**, escoger el **General** elemento, y luego **Accesibilidad**, y

Texto más grande. Una barra de desplazamiento está disponible para hacer el texto más pequeño o más grande. Cambiar eso deslizante, doble toque la

Casa botón para mostrar las aplicaciones actuales, y seleccione **DeviceStyles** de nuevo. Verá el texto que figura entre los estilos de dispositivo (o los estilos que se derivan de estilos de dispositivos) cambian de tamaño, pero ninguna parte del texto sin estilo en el tamaño de cambios en las aplicaciones. Los nuevos objetos han sustituido a los estilos de dispositivos en el diccionario.

Tamaño de fuente elemento de la Monitor en la sección ajustes afectará a todos los tamaños de fuente en un programa Xamarin.Forms.

En un dispositivo móvil de Windows 10, el **Ajuste de texto** elemento de la **Facilidad de Acceso y Mas opciones**

Sección de ajustes También afecta a todo el texto

El siguiente capítulo se incluye un programa que muestra cómo hacer un pequeño lector de libros electrónicos que le permite leer un capítulo de *Alicia en el país de las Maravillas*. Este programa utiliza estilos de dispositivos para controlar el formato de todo el texto, incluyendo los títulos de los libros y de capítulos.

Pero lo que este pequeño lector de libros electrónicos son también incluye ilustraciones, y que requiere una exploración en el tema de los mapas de bits.

capítulo 13

Los mapas de bits

Los elementos visuales de una interfaz gráfica de usuario se pueden dividir entre los elementos utilizados para la presentación (como texto) y aquellos capaces de interacción con el usuario, tales como botones, barras de desplazamiento, y cuadros de lista.

El texto es esencial para la presentación, pero las imágenes son a menudo tan importante como una forma de complementar el texto y transmitir información crucial. La web, por ejemplo, sería inconcebible sin imágenes. Estas imágenes son a menudo en forma de matrices rectangulares de elementos de imagen (o píxeles) conocidos como *mapas de bits*.

Al igual que una vista denominada **Etiqueta pantallas de texto**, una vista llamada **Imagen muestra mapas de bits**. Los formatos de mapa de bits con el apoyo de iOS, Android y el tiempo de ejecución de Windows son un poco diferentes, pero si usted se pega a JPEG, PNG, GIF, BMP y en sus aplicaciones Xamarin.Forms, es probable que no experimentan ningún problema.

Imagen define una **Fuente** propiedad que se establece a un objeto de tipo **Fuente de imagen**, que hace referencia al mapa de bits mostrado por **Imagen**. Los mapas de bits pueden venir de una variedad de fuentes, por lo que la **Fuente de imagen** clase define cuatro métodos de creación estáticos que devuelven una **Fuente de imagen** objeto:

- **ImageSource.FromUri** para acceder a un mapa de bits a través de Internet.
- **ImageSource.FromResource** para un mapa de bits almacenado como un recurso incrustado en el PCL aplicación.
- **ImageSourceFromFile** para un mapa de bits almacenado como contenido en un proyecto de plataforma individual.
- **ImageSource.FromStream** para cargar un mapa de bits utilizando un .NET Corriente objeto.

Fuente de imagen También tiene tres clases descendientes, llamados **UriImageSource**, **FileImageSource**, y **StreamImageSource**, que se puede utilizar en lugar de la primera, tercera, y cuarta métodos de creación estáticos. En general, los métodos estáticos son más fáciles de usar en el código, pero las clases descendientes veces se requieren en XAML.

En general, va a utilizar el **ImageSource.FromUri** y **ImageSource.FromResource** métodos para obtener mapas de bits independientes de la plataforma con fines de presentación y **ImageSource.FromFile** para cargar mapas de bits específicos de la plataforma para los objetos de interfaz de usuario. Mapas de bits pequeños juegan un papel crucial en la Opción del menú y **ToolbarItem** Los objetos, y también se puede añadir un mapa de bits a una **Botón**.

Este capítulo comienza con el uso de mapas de bits independientes de la plataforma obtenida de la **ImageSource.FromUri** y **ImageSource.FromResource** métodos. A continuación, explora algunos usos de la **ImageSource.FromStream** método. El capítulo concluye con el uso de **ImageSourceFromFile** para obtener mapas de bits específicos de la plataforma para las barras de herramientas y botones.

mapas de bits independientes de la plataforma

He aquí un programa de sólo código llamado **WebBitmapCode** con una clase de página que utiliza `Imagen-Source.FromUri` acceder a un mapa de bits de la página web Xamarin:

```
p\xedblico clase WebBitmapCodePage : Pagina de contenido
{
    p\xedblico WebBitmapCodePage ()
    {
        cuerda uri = "Https://developer.xamarin.com/demo/IMG_1415.JPG";

        content = nuevo Imagen
        {
            fuente = Fuente de imagen .FromUri ( nuevo Uri (URI));
        };
    }
}
```

Si el URI transferido `ImageSource.FromUri` no apunta a un mapa de bits válidos, se eleva no es una excepción.

Incluso este pequeño programa se puede simplificar. `Fuente de imagen` define una conversión implícita de `cuerda Uri` a una `Fuente de imagen` oponerse, lo que puede establecer la cadena con el URI directamente a la `Fuente` propiedad de `Imagen`:

```
p\xedblico clase WebBitmapCodePage : Pagina de contenido
{
    p\xidblico WebBitmapCodePage ()
    {
        content = nuevo Imagen
        {
            fuente = "Https://developer.xamarin.com/demo/IMG_1415.JPG";
        };
    }
}
```

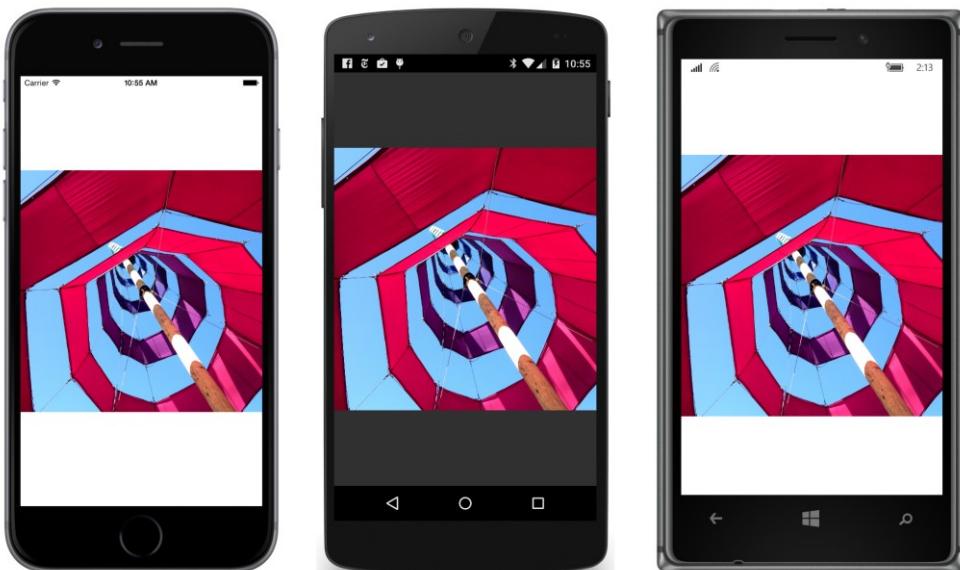
O bien, para que sea más detallado, se puede establecer el `Fuente` propiedad de `Imagen` a una `UriImageSource` objeto con su `Uri` propiedad establecida en una `Uri` objeto:

```
p\xidblico clase WebBitmapCodePage : Pagina de contenido
{
    p\xidblico WebBitmapCodePage ()
    {
        content = nuevo Imagen
        {
            fuente = nuevo UriImageSource
            {
                uri = nuevo Uri ("Https://developer.xamarin.com/demo/IMG_1415.JPG");
            };
        };
    }
}
```

los UriImageSource clase puede ser preferible si se quiere controlar el almacenamiento en caché de imágenes basadas en la Web. La clase implementa su propio almacenamiento en caché que utiliza el área de almacenamiento privado de la aplicación disponible en cada plataforma. UriImageSource define una CachingEnabled propiedad que tiene un valor predeterminado de

cierto y una CachingValidity propiedad de tipo Espacio de tiempo que tiene un valor predeterminado de un día. Esto significa que si la imagen se reaccessed dentro de un día, se utilizará la imagen almacenada en caché. Puede deshabilitar el almacenamiento en caché en su totalidad mediante el establecimiento CachingEnabled a falso, o puede cambiar el tiempo de caducidad de almacenamiento en caché mediante el establecimiento de la CachingValidity propiedad a otra Espacio de tiempo valor.

Independentemente de qué manera lo hace, por defecto, el mapa de bits mostrado por el Imagen vista se estira con el tamaño de su contenedor-la Pagina de contenido en este caso-respetando al mismo tiempo la relación de aspecto del mapa de bits:

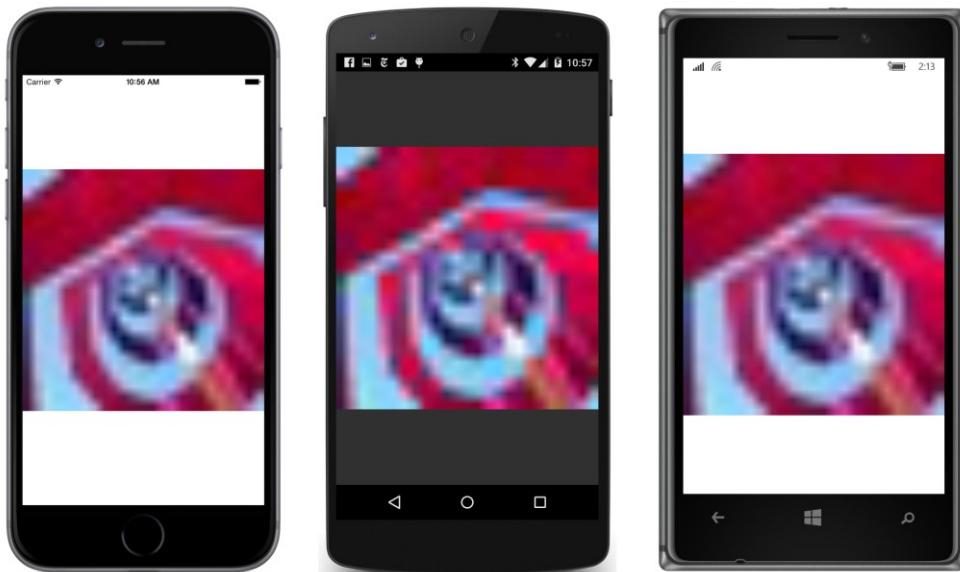


Este mapa de bits es cuadrada, por lo que las áreas en blanco aparecen por encima y por debajo de la imagen. A medida que gira el teléfono o emulador entre el modo vertical y horizontal, un mapa de bits prestados puede cambiar el tamaño, y verá algo de espacio en blanco en la parte superior e inferior o a la izquierda ya la derecha, en el que el mapa de bits no llega. Puede color esa zona mediante el uso de la Color de fondo propiedad que Imagen hereda de VisualElement.

El mapa de bits que se hace referencia en el [WebBitmapCode](#) programa es de 4.096 pixeles cuadrados, sino una utilidad se instala en el sitio web Xamarin que le permite descargar un archivo de mapa de bits mucho más pequeños especificando el URI de este modo:

```
content = nuevo Imagen
{
    fuente = "Https://developer.xamarin.com/demo/IMG_1415.JPG?width=25"
};
```

Ahora el mapa de bits descargado es de 25 píxeles cuadrados, pero se estira de nuevo al tamaño de su contenedor. Cada plataforma implementa un algoritmo de interpolación en un intento de suavizar los píxeles que la imagen se expande para ajustarse a la página:



Sin embargo, si se establece ahora `HorizontalOptions` y `VerticalOptions` sobre el `Imagen` a Centrar - o poner el `Imagen` elemento en una `StackLayout` -Este mapa de bits de 25 píxeles colapsa en una imagen muy pequeña. Este fenómeno se explica con más detalle más adelante en este capítulo.

También puede crear una instancia de una `Imagen` elemento en XAML y cargar un mapa de bits de una dirección URL mediante el establecimiento de la `Fuente` la propiedad directamente a una dirección web. Aquí está el archivo XAML de la `WebBitmapXaml` programa:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns:x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " WebBitmapXaml.WebBitmapXamlPage " >

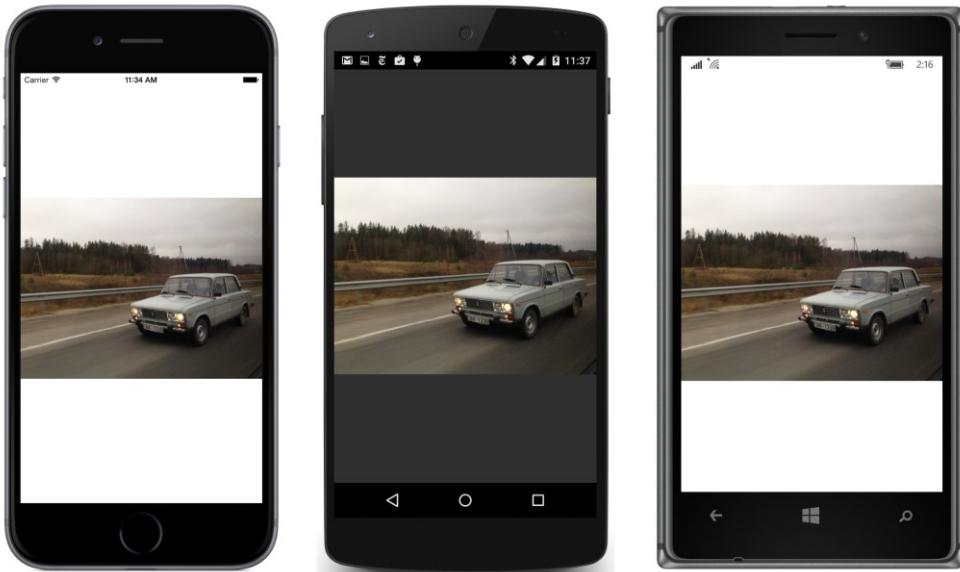
    < Imagen Fuente = " https://developer.xamarin.com/demo/IMG_3256.JPG " />

</ Página de contenido >
```

Un enfoque más detallado implica crear instancias explícitamente una `UriImageSource` objeto y establecer el `Uri` propiedad:

```
< Imagen >
    < Fuente de imagen >
        < UriImageSource Uri = " https://developer.xamarin.com/demo/IMG_3256.JPG " />
    </ Fuente de imagen >
</ Imagen >
```

De todos modos, aquí está cómo se ve en la pantalla:



Montar y llenar

Si se establece el **Color de fondo** propiedad de **Imagen** en cualquiera de los ejemplos de código y XAML anteriores, verá que **Imagen** en realidad ocupa toda el área rectangular de la página. **Imagen** define una **Comportamiento** propiedad que controla la forma del mapa de bits se dictará dentro de este rectángulo. Se establece esta propiedad a un miembro de la **Aspecto** enumeración:

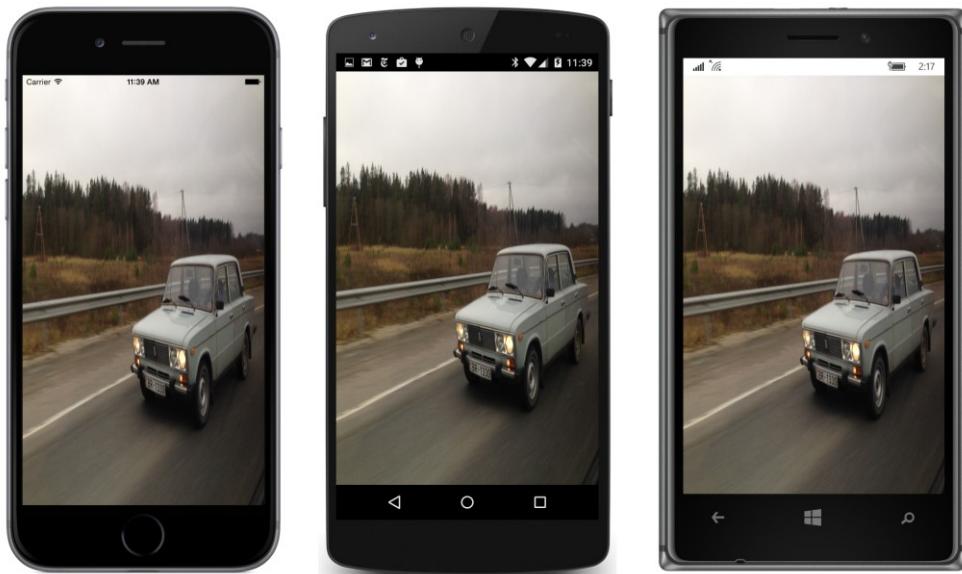
- **AspectFit** - el valor por defecto
- **Llenar** - se extiende sin conservar la relación de aspecto
- **AspectFill** - conserva la relación de aspecto, pero recorta la imagen

La configuración por defecto es el miembro de la enumeración **Aspect AspectFit**, lo que significa que el mapa de bits se ajusta a los límites de su contenedor preservando al mismo tiempo la relación de aspecto del mapa de bits. Como ya se ha visto, la relación entre las dimensiones del mapa de bits y las dimensiones del envase puede resultar en áreas de fondo en la parte superior y la parte inferior o en la derecha y la izquierda.

Pruebe esto en el **WebBitmapXaml** proyecto:

```
<Imagen Fuente = "https://developer.xamarin.com/demo/IMG_3256.JPG">
    Aspecto = "Llenar" />
```

Ahora el mapa de bits se expande a las dimensiones de la página. Esto se traduce en la imagen que se extiende verticalmente, así que el coche parece más bien bajo y robusto:

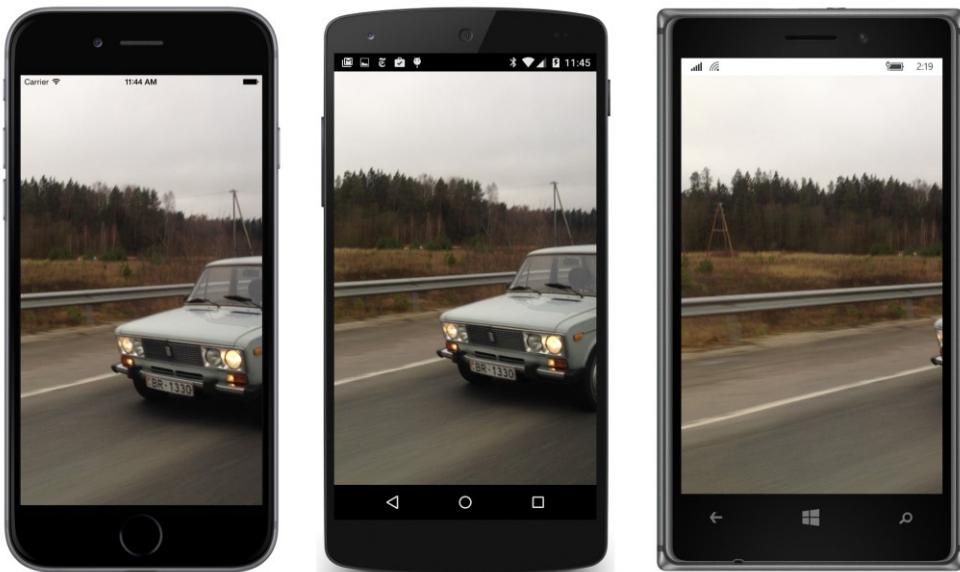


Si se gira el teléfono hacia los lados, la imagen se estira horizontalmente, pero el resultado no es tan extrema, porque la relación de aspecto de la imagen es un tanto paisaje, para empezar.

La tercera opción es `AspectFill`:

```
<Imagen Fuente = "https://developer.xamarin.com/demo/IMG\_3256.JPG"  
Aspecto = "AspectFill" />
```

Con esta opción, el mapa de bits llena completamente el recipiente, pero la relación de aspecto del mapa de bits se mantiene al mismo tiempo. La única forma en que esto es posible es recortando parte de la imagen, y verá que la imagen está recortada en efecto, pero de una manera diferente en las tres plataformas. En iOS y Android, se recorta la imagen en la parte superior e inferior o en la izquierda y la derecha, dejando sólo la parte central del mapa de bits visible. En las plataformas de Windows en tiempo de ejecución, se recorta la imagen a la derecha o abajo, dejando la esquina superior izquierda visibles:



recursos incrustados

Acceso a mapas de bits a través de Internet es conveniente, pero a veces no es óptima. El proceso requiere una conexión a Internet, una garantía de que los mapas de bits no se han movido, y algo de tiempo para su descarga. Para un acceso rápido y garantizado a los mapas de bits, que se pueden unir a la derecha en la aplicación.

Si necesita acceder a las imágenes que no son específicos de la plataforma, puede incluir mapas de bits como recursos incrustados en el proyecto de biblioteca compartida Clase portátil y acceder a ellos con la `ImageSource.FromResource` método. la **ResourceBitmapCode** solución demuestra cómo hacerlo.

los **ResourceBitmapCode** proyecto PCL dentro de esta solución tiene una carpeta llamada **Imágenes** que contiene dos mapas de bits, el nombre `ModernUserInterface.jpg` (un muy gran mapa de bits) y `ModernUserInterface256.jpg` (la misma imagen, pero con una anchura de 256 pixeles).

Al añadir cualquier tipo de recurso incrustado a un proyecto PCL, asegúrese de establecer la **construir Acción** del recurso a **EmbeddedResource**. Esto es crucial.

En el código, se establece la **Fuente** propiedad de una **Imagen** elemento a la **Fuente de imagen** objeto volvió de la estática `ImageSource.FromResource` método. Este método requiere el ID de recurso. El ID de recurso consiste en el nombre de ensamblado seguido de un período, entonces el nombre de la carpeta seguido por otro período, y después el nombre de archivo, que contiene otro período para la extensión de nombre de archivo. Para este ejemplo, el ID de recurso para acceder al más pequeño de los dos mapas de bits en el **ResourceBitmapCode**

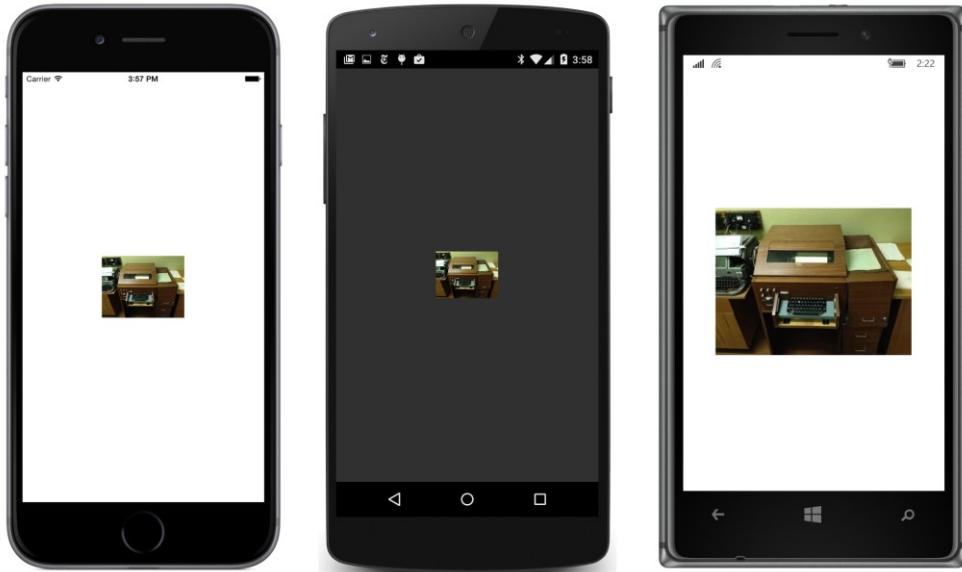
programa es:

```
ResourceBitmapCode.Images.ModernUserInterface256.jpg
```

El código de este programa hace referencia a ese mapa de bits más pequeño y también establece el `HorizontalOptions` y `VerticalOptions` sobre el `Imagen` elemento de Centrar:

```
p\u00f3blico clase ResourceBitmapCodePage : Página de contenido
{
    p\u00f3blico ResourceBitmapCodePage ()
    {
        content = nuevo Imagen
        {
            fuente = Fuente de imagen .FromResource (
                "ResourceBitmapCode.Images.ModernUserInterface256.jpg" ),
            VerticalOptions = LayoutOptions .Centrar,
            HorizontalOptions = LayoutOptions .Centrar
        };
    }
}
```

Como se puede ver, el mapa de bits en este caso es *no* estirada para llenar la página:



Un mapa de bits no se estira para llenar su contenedor si:

- es más pequeño que el recipiente, y
- el `VerticalOptions` y `HorizontalOptions` propiedades de la `Imagen` elemento que no se ajustan a Llenar, o si `Imagen` es un hijo de una `StackLayout`.

Si en comentario la `VerticalOptions` y `HorizontalOptions` configuración, o si hace referencia a la gran mapa de bits (que no tiene el "256" al final de su nombre de archivo), la imagen volverá a estirar para llenar el recipiente.

Cuando un mapa de bits no se estira para adaptarse a su contenedor, debe mostrarse en un tamaño determinado. ¿Qué es de ese tamaño?

En iOS y Android, el mapa de bits se muestra en su tamaño en píxeles. En otras palabras, el mapa de bits se representa con una asignación uno a uno entre los píxeles del mapa de bits y los píxeles de la pantalla de video. El 6 simulador iPhone utilizado para estas capturas de pantalla tiene una anchura de la pantalla de 750 píxeles, y se puede ver que la anchura de 256 píxeles del mapa de bits es de aproximadamente un tercio de la anchura. El teléfono Android que aquí hay una Nexus 5, que tiene un ancho de pixel de 1080, y el mapa de bits es de aproximadamente un cuarto de la anchura.

En las plataformas de Windows en tiempo de ejecución, sin embargo, el mapa de bits se muestra en unidades en este ejemplo, las unidades 256 de dispositivo independientes del dispositivo. El Nokia Lumia 925 se utiliza para estas capturas de pantalla tiene un ancho de píxel de 768, que es aproximadamente el mismo que el iPhone 6. Sin embargo, el ancho de la pantalla de este Windows 10 Teléfono móvil en unidades independientes del dispositivo es de 341, y se puede ver que el mapa de bits prestados es mucho más amplio que en las otras plataformas.

Esta discusión sobre los mapas de bits de tamaño continúa en la siguiente sección.

¿Cómo se hace referencia a un mapa de bits almacenado como un recurso incrustado de XAML? Desafortunadamente no hay `ResourceImageSource` clase. Si lo hubiera, es probable que intenta crear instancias de esa clase en XAML entre Fuente de imagen las etiquetas. Pero eso no es una opción.

Le recomendamos que utilice `x: FactoryMethod` llamar `ImageSource.FromResource`, pero eso no va a funcionar. Tal como se aplica actualmente, el `ImageSource.FromResource` método requiere que el recurso de mapa de bits sea en el mismo conjunto como el código que llama al método. Cuando se utiliza `x: FactoryMethod`

llamar `ImageSource.FromResource`, la llamada se realiza desde el **Xamarin.Forms.Xaml** montaje.

Qué *será* el trabajo es muy simple extensión de marcado XAML. Aquí está uno en un proyecto llamado **StackedBitmap**:

```
espacio de nombres StackedBitmap
{
    [ ContentProperty ( "Fuente" ) ]
    clase pública ImageResourceExtension : IMarkupExtension
    {
        public string Fuente { obtener ; conjunto ; }

        objeto público ProvideValue ( IServiceProvider proveedor de servicio)
        {
            Si (Fuente == nulo )
                return null ;

            regreso Fuente de imagen .FromResource (Fuente);
        }
    }
}
```

`ImageResourceExtension` tiene una sola propiedad llamada `Fuente` que se establece en el ID de recurso. los

`ProvideValue` método llama simplemente `ImageSource.FromResource` con el `Fuente` propiedad. Como es común para las extensiones de marcado de una sola propiedad, `Fuente` es también la propiedad de contenido de la clase. Ese

significa que no es necesario incluir de forma explícita "Fuente =" cuando se está utilizando la sintaxis entre llaves para extensiones de marcado XAML.

Pero tenga cuidado: No se puede mover este ImageResourceExtension clase a una biblioteca tal como Xamarin.FormsBook.Toolkit. La clase debe ser parte del mismo conjunto que contiene los recursos incrustados que desea cargar, que es generalmente de biblioteca de clases portátil de la aplicación.

Aquí está el archivo XAML de la StackedBitmap proyecto. Un Imagen comparte un elemento StackLayout con dos Etiqueta elementos:

```
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    xmlns: locales = " CLR-espacio de nombres: StackedBitmap "
    x: Class = " StackedBitmap.StackedBitmapPage " >

    < StackLayout >
        < Etiqueta Texto = " Mapa de bits 320 x 240 píxeles "
            Tamaño de fuente = " Medio "
            VerticalOptions = " CenterAndExpand "
            HorizontalOptions = " Centrar " />

        < Imagen Fuente = " {Locales: ImageResource StackedBitmap.Images.Sculpture_320x240.jpg} "
            Color de fondo = " Agua "
            SizeChanged = " OnImageSizeChanged " />

        < Etiqueta x: Nombre = " etiqueta "
            Tamaño de fuente = " Medio "
            VerticalOptions = " CenterAndExpand "
            HorizontalOptions = " Centrar " />
    </ StackLayout >
</ Pagina de contenido >
```

los local prefijo se refiere a la StackedBitmap en el espacio de nombres StackedBitmap montaje. los Fuente propiedad de la Imagen elemento se establece en el ImageResource extensión de marcado, que hace referencia a un mapa de bits almacenado en el Imágenes carpeta del proyecto PCL y marcado como una EmbeddedResource.

El mapa de bits es de 320 píxeles de ancho y 240 píxeles de alto. los Imagen También tiene su Color de fondo conjunto de propiedades; esto nos permitirá ver todo el tamaño Imagen dentro de StackLayout.

los Imagen elemento tiene su SizeChanged evento que se desarrolla a un controlador en el archivo de código subyacente:

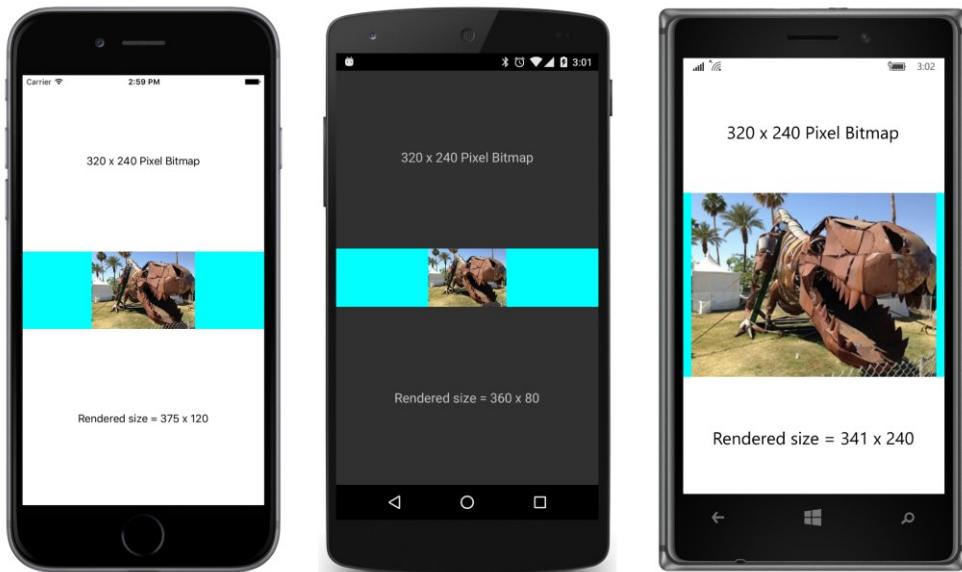
```
público clase parcial StackedBitmapPage : Pagina de contenido
{
    público StackedBitmapPage ()
    {
        InitializeComponent ();
    }

    vacío OnImageSizeChanged ( objeto remitente, EventArgs args )
    {
        Imagen image = ( Imagen )remitente;
        label.text = Cuerda .Formato( "Render size = {0: F0} x {1: F0}" ,
            image.Width, image.Height);
    }
}
```

```
    }  
}
```

El tamaño de la Imagen elemento está limitado verticalmente por el StackLayout, por lo que el mapa de bits se muestra en su tamaño en píxeles (en iOS y Android) y en unidades independientes del dispositivo en Windows Phone. los

Etiqueta muestra el tamaño de la Imagen elemento en unidades independientes del dispositivo, que difieren en cada plataforma:

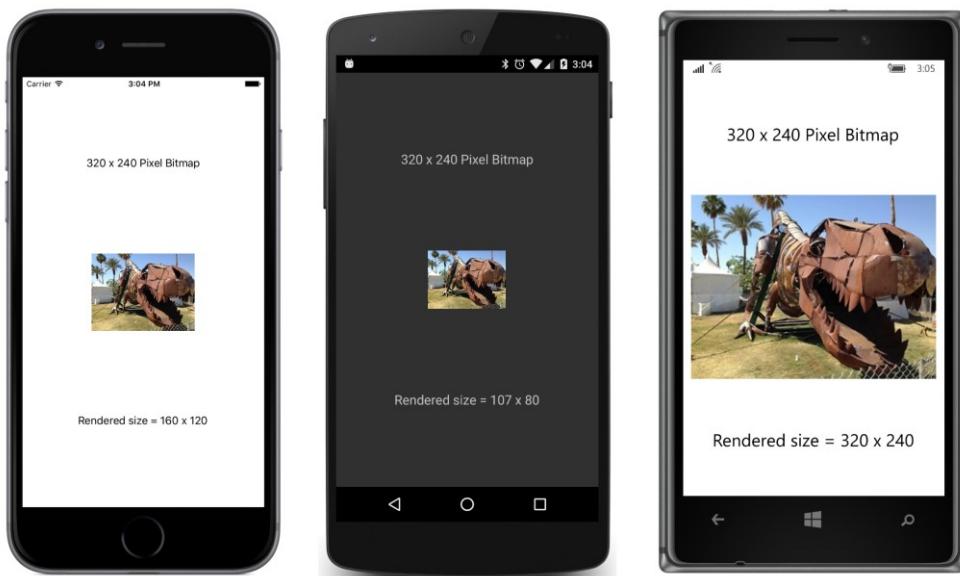


La anchura de la Imagen elemento mostrado por la parte inferior Etiqueta incluye el fondo de la aguamarina y es igual al ancho de la página en unidades independientes del dispositivo. Puedes usar Aspecto ajustes de Llenar o Como-pectFill para hacer que el mapa de bits que llena toda la zona Aqua.

Si prefiere que el tamaño de la Imagen elemento sea del mismo tamaño que el mapa de bits independiente del dispositivo dictada en unidades, se puede establecer el HorizontalOptions propiedad de la Imagen a algo que no sea el valor por defecto de Llenar:

```
<Imagen Fuente = " {Locales: ImageResource StackedBitmap.Images.Sculpture_320x240.jpg } "  
    HorizontalOptions = "Centrar"  
    Color de fondo = "Aqua"  
    SizeChanged = "OnImageSizeChanged" />
```

Ahora la parte inferior Etiqueta muestra sólo el ancho del mapa de bits prestados. Ajustes de la Aspecto la propiedad tiene ningún efecto:



Vamos a referirnos a este prestados Imagen tamaño que su *tamaño natural*, ya que se basa en el tamaño del mapa de bits que se muestra.

El iPhone 6 tiene un ancho de píxel de 750 píxeles, pero como se descubrió cuando se ejecuta el **Qué tamaño** programa en el Capítulo 5, las aplicaciones percibir un ancho de pantalla de 375. Hay dos píxeles a la unidad independiente del dispositivo, por lo que un mapa de bits con un ancho de 320 píxeles se muestra con una anchura de 160 unidades.

El Nexus 5 tiene una anchura de pixel de 1080, pero las aplicaciones percibir una anchura de 360, por lo que hay tres píxeles a la unidad independiente del dispositivo, como el **Imagen anchura de 107 unidades** confirma.

En ambas iOS y dispositivos Android, cuando se muestra un mapa de bits en su tamaño natural, hay una asignación uno Toone entre los píxeles del mapa de bits y los píxeles de la pantalla. En los dispositivos de Windows en tiempo de ejecución, sin embargo, que no es el caso. El Nokia Lumia 925 se utiliza para estas capturas de pantalla tiene un ancho de pixel de 768. Cuando se ejecuta el sistema operativo Windows Mobile 10, hay 2,25 píxeles a la unidad independiente del dispositivo, por lo que las aplicaciones perciben un ancho de pantalla de 341. Sin embargo, el mapa de bits de 320 x 240 píxeles se visualiza en un tamaño de 320 x 240 independiente del dispositivo unidades.

Esta inconsistencia entre el tiempo de ejecución de Windows y las otras dos plataformas es realmente beneficioso cuando se está accediendo a los mapas de bits a partir de los proyectos de plataforma individuales. Como verá, el IOS y Android incluyen una característica que le permite abastecer a los diferentes tamaños de mapas de bits para diferentes resoluciones de dispositivos. En efecto, esto le permite especificar tamaños de mapa de bits en unidades independientes del dispositivo, lo que significa que los dispositivos de Windows son compatibles con esos esquemas.

Sin embargo, cuando se usa mapas de bits independientes de la plataforma, es probable que desee tamaño de los mapas de bits de forma consistente en las tres plataformas, y que requiere un paso más en el tema.

Más sobre dimensionamiento

Hasta ahora, usted ha visto dos formas de tamaño Imagen elementos:

Si el Imagen elemento no está limitado de ninguna manera, llenará su recipiente mientras se mantiene la relación de aspecto del mapa de bits, o rellenar el área por completo si se establece la Aspecto propiedad a Llenar o AspectFill.

Si el mapa de bits es menor que el tamaño de su contenedor y la Imagen está restringida horizontal o vertical mediante el establecimiento HorizontalOptions o VerticalOptions a algo distinto Llenar, o si el Imagen se pone en una StackLayout, el mapa de bits se muestra en su tamaño natural. Ese es el tamaño de pixel en iOS y Android, pero el tamaño en unidades independientes del dispositivo en dispositivos Windows.

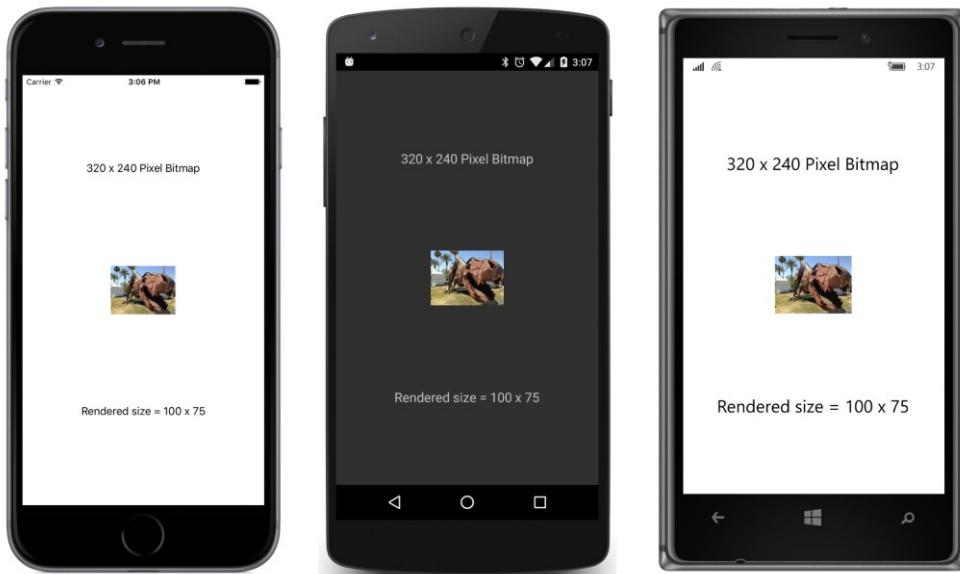
También puede controlar el tamaño mediante el establecimiento WidthRequest o HeightRequest a una dimensión explícita en unidades independientes del dispositivo. Sin embargo, hay algunas restricciones.

La siguiente discusión se basa en la experimentación con el StackedBitmap muestra. Que se refiere a Imagen elementos que están restringidas verticalmente por ser un hijo de una vertical de StackLayout o que tengan el VerticalOptions propiedad establecida en algo que no sea Llenar. Los mismos principios se aplican a una Imagen elemento que está constreñido horizontalmente.

Si una Imagen elemento está limitado verticalmente, puede utilizar WidthRequest para reducir el tamaño del mapa de bits de su tamaño natural, pero no se puede utilizar para aumentar el tamaño. Por ejemplo, intente establecer WidthRequest a 100:

```
<Imagen Fuente = " (Locales: ImageResource StackedBitmap.Images.Sculpture_320x240.jpg ) "
    WidthRequest = " 100 "
    HorizontalOptions = " Centrar "
    Color de fondo = " Agua "
    SizeChanged = " OnImageSizeChanged " />
```

La altura resultante del mapa de bits se rige por la anchura especificada y relación de aspecto del mapa de bits, por lo que ahora el Imagen se muestra con un tamaño de 100 x 75 unidades independientes del dispositivo en las tres plataformas:

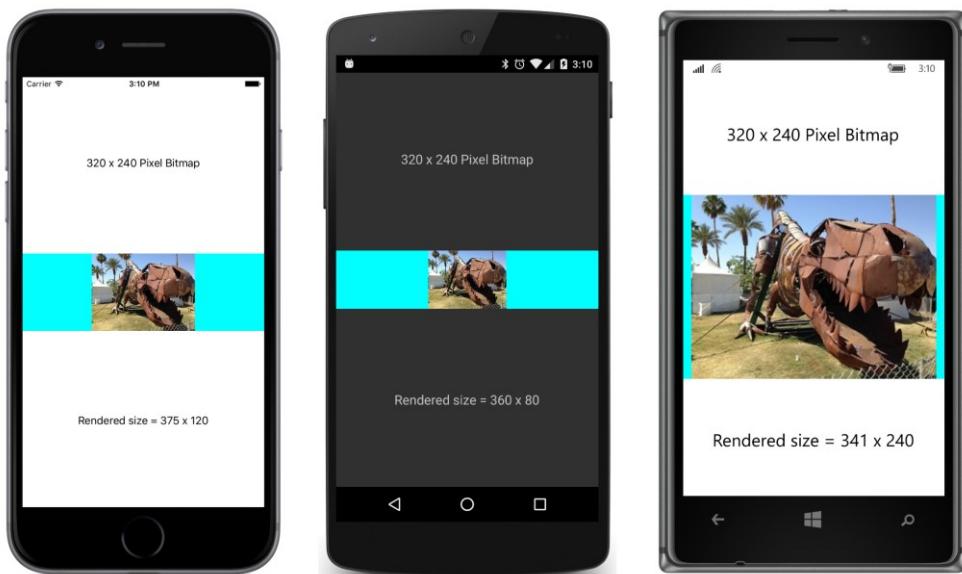


los `HorizontalOptions` ajuste de `Centrar` no afecta el tamaño del mapa de bits prestados. Si se quita esa línea, la `Imagen` elemento será tan ancha como la pantalla (como el color de fondo aqua demostrará), pero el mapa de bits permanecerá el mismo tamaño.

No puede utilizar `WidthRequest` para aumentar el tamaño del mapa de bits prestados más allá de su tamaño natural. Por ejemplo, intente establecer `WidthRequest` a 1000:

```
<Imagen Fuente = "{Locales: ImageResource StackedBitmap.Images.Sculpture_320x240.jpg}"*
    WidthRequest = "1000"
    HorizontalOptions = "Centrar"
    Color de fondo = "Agua"
    SizeChanged = "OnImageSizeChanged" />
```

Incluso con `HorizontalOptions` ajustado a `Centrar`, el resultante `Imagen` elemento es ahora más ancho que el mapa de bits prestados, según lo indicado por el color de fondo:



Pero el mapa de bits en sí está representada en su tamaño natural. la vertical StackLayout impide eficazmente la altura del mapa de bits representan a partir superior a su altura natural.

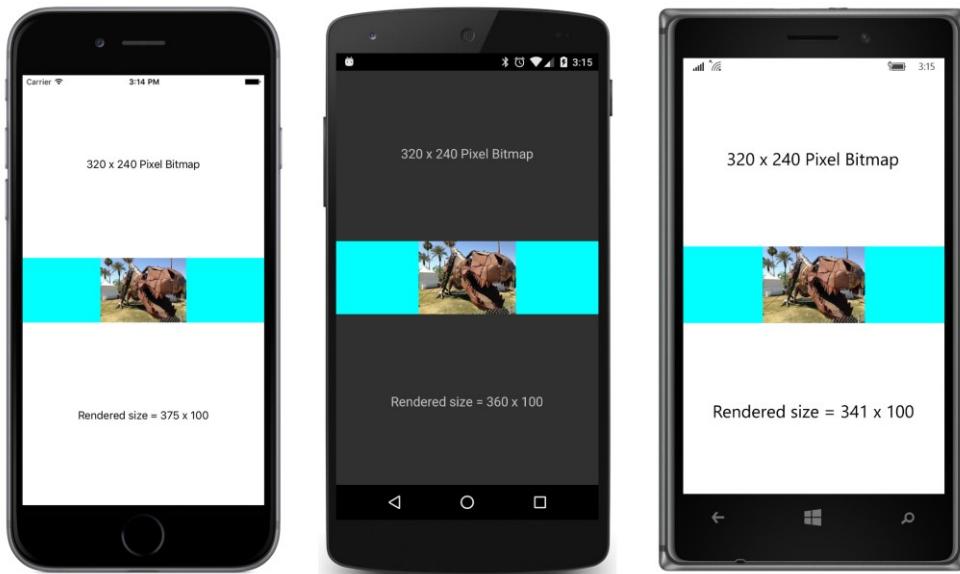
Para superar esa limitación de la vertical, StackLayout, es necesario establecer HeightRequest. Sin embargo, también querrá salir HorizontalOptions en su valor por defecto de Llenar. De lo contrario, el horizontalOptions ajuste evitará que el ancho del mapa de bits prestados de superior a su tamaño natural.

Al igual que con WidthRequest, se puede establecer HeightRequest para reducir el tamaño del mapa de bits prestados. Los siguientes conjuntos de códigos HeightRequest a 100 unidades independientes del dispositivo:

```
<Imagen Fuente = " (Locales: ImageResource StackedBitmap.Images.Sculpture_320x240.jpg ) "
    HeightRequest = " 100 "
    Color de fondo = " Agua "
    SizeChanged = " OnImageSizeChanged " />
```

Observe también que el HorizontalOptions ajuste se ha eliminado.

El mapa de bits mostrada es ahora 100 unidades independientes del dispositivo de alta con un ancho gobernado por la relación de aspecto. los Imagen propio elemento se extiende a los lados de la StackLayout:

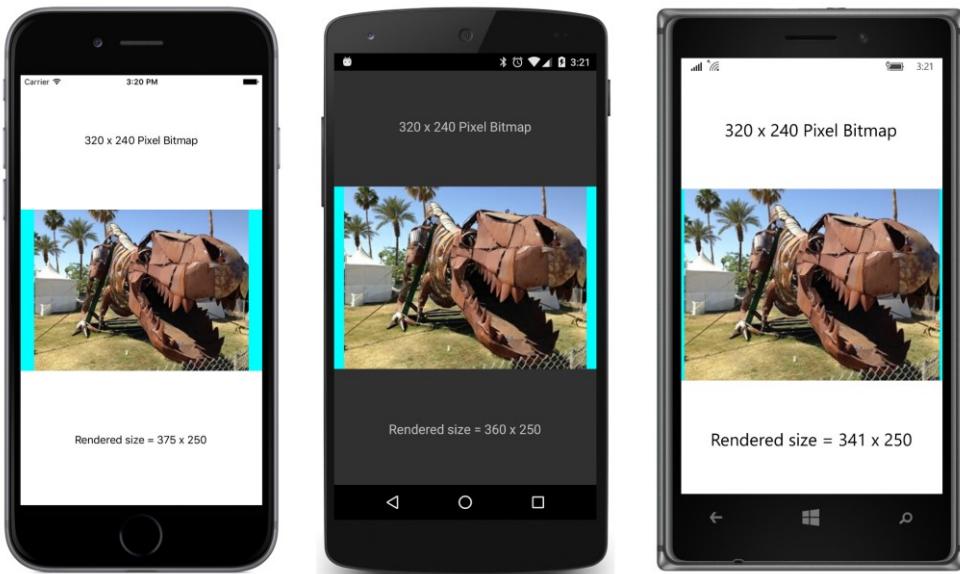


En este caso particular, se puede establecer `HorizontalOptions` a `Centrar` sin cambiar el tamaño del mapa de bits prestados. Los `Imagen` elemento será entonces el tamaño del mapa de bits (133 x 100), y el fondo de la aguamarina desaparecerá.

Es importante dejar `HorizontalOptions` en su configuración por defecto de `Llenar` cuando se ajusta el `HeightRequest` a un valor mayor que la altura natural del mapa de bits, por ejemplo 250:

```
<Imagen Fuente = "{Locales: ImageResource StackedBitmap.Images.Sculpture_320x240.jpg}"  
      HeightRequest = "250"  
      Color de fondo = "Agua"  
      SizeChanged = "OnImageSizeChanged" />
```

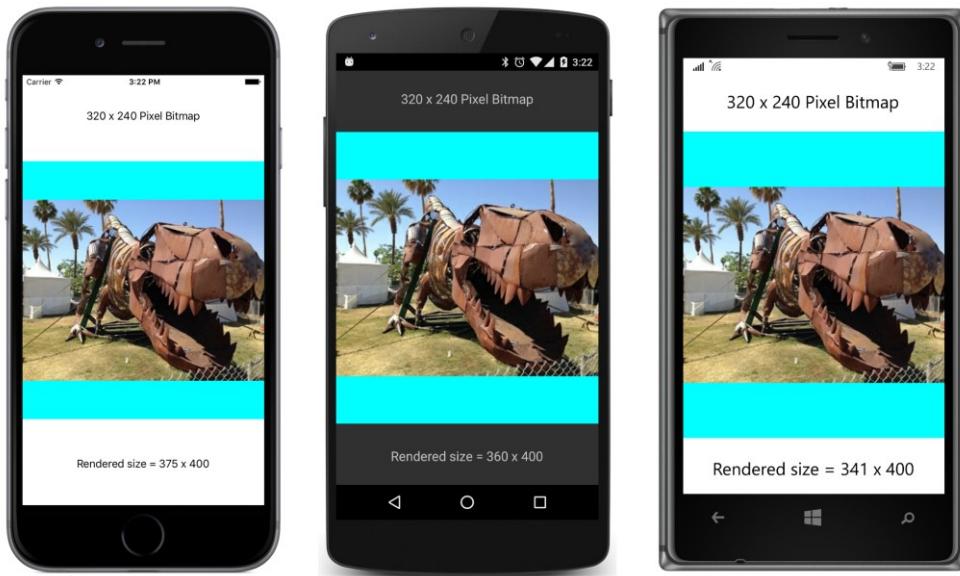
Ahora el mapa de bits prestados es más grande que su tamaño natural:



Sin embargo, esta técnica tiene un sistema incorporado en peligro, que se revela cuando se establece el HeightRequest a 400:

```
<Imagen Fuente = "Locales: ImageResource StackedBitmap.Images.Sculpture_320x240.jpg"
    HeightRequest = "400"
    Color de fondo = "Aqua"
    SizeChanged = "OnImageSizeChanged" />
```

Esto es lo que sucede: El Imagen elemento en efecto, obtener una altura de 400 unidades independientes del dispositivo. Pero el ancho del mapa de bits dictado en dicha Imagen elemento está limitado por el ancho de la pantalla, lo que significa que la altura del mapa de bits mostrada es menor que la altura de la Imagen elemento:



En un programa real que probablemente no tienen la Color de fondo conjunto de propiedades, y en lugar de un terreno baldío de la pantalla en blanco ocupará el área en la parte superior y la parte inferior del mapa de bits prestados.

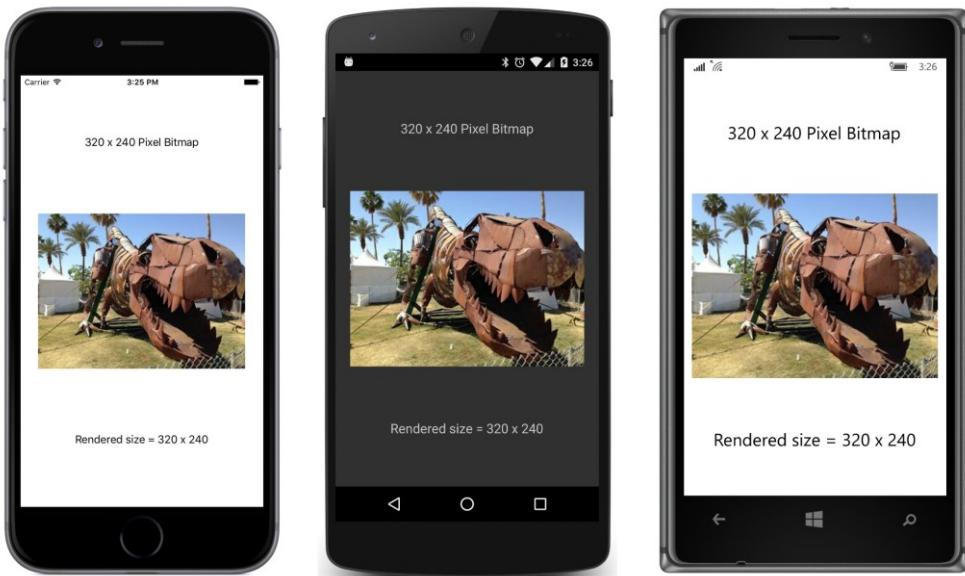
Lo que esto implica es que no se debe utilizar HeightRequest para controlar el tamaño de los mapas de bits en una vertical de StackLayout a menos que escribir código que garantiza que HeightRequest se limita a la anchura de la StackLayout veces la relación de la altura del mapa de bits a la anchura.

Si se conoce el tamaño de píxel del mapa de bits que se le vaya a mostrar, un método fácil es establecer

WidthRequest y HeightRequest a ese tamaño:

```
<Imagen Fuente = " {Locales: ImageResource StackedBitmap.Images.Sculpture_320x240.jpg } "
    WidthRequest = " 320 "
    HeightRequest = " 240 "
    HorizontalOptions = " Centrar "
    Color de fondo = " Agua "
    SizeChanged = " OnImageSizeChanged " />
```

Ahora el mapa de bits se muestra en ese tamaño en unidades independientes del dispositivo en todas las plataformas:



El problema aquí es que el mapa de bits no se está mostrando en su resolución óptima. Cada píxel del mapa de bits ocupa al menos dos píxeles de la pantalla, dependiendo del dispositivo.

Si desea mapas de bits de tamaño de forma vertical `StackLayout` para que se vieran aproximadamente el mismo tamaño en una variedad de dispositivos, el uso `WidthRequest` más bien que `HeightRequest`. Usted ha visto que `WidthRequest` en una vertical de `StackLayout` sólo puede disminuir el tamaño de los mapas de bits. Esto significa que se debe utilizar mapas de bits que son más grandes que el tamaño en el que van a ser prestados. Esto le dará una solución más óptima cuando la imagen tiene un tamaño en unidades independientes del dispositivo. Puede cambiar el tamaño del mapa de bits mediante el uso de un tamaño métrico deseado en pulgadas junto con el número de unidades independientes del dispositivo a la pulgada para el dispositivo particular, el cual nos pareció ser 160 para estos tres dispositivos.

Aquí hay un proyecto muy similar a `StackedBitmap` llamado `DeviceIndBitmapSize`. Es el mismo mapa de bits, pero ahora 1200×900 píxeles, lo que es más ancho que el ancho en modo retrato de incluso de alta resolución de 1920×1080 pantallas. La anchura solicitada específica de la plataforma del mapa de bits corresponde a 1,5 pulgadas:

```
< Página de contenido xmlns = " http://kamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    xmlns: locales = " CLR-espacio de nombres: DeviceIndBitmapSize "
    x: Class = " DeviceIndBitmapSize.DeviceIndBitmapSizePage " >

    < StackLayout >
        < Etiqueta Texto = " 1200 x 900 píxeles Bitmap "
            Tamaño de fuente = " Medio "
            VerticalOptions = " CenterAndExpand "
            HorizontalOptions = " Centrar " />

        <! - ancho de la imagen de 1,5 pulgadas ->
        < Imagen Fuente = " (Locales: ImageResource DeviceIndBitmapSize.Images.Sculpture_1200x900.jpg) "
            WidthRequest = " 240 " >
```

```

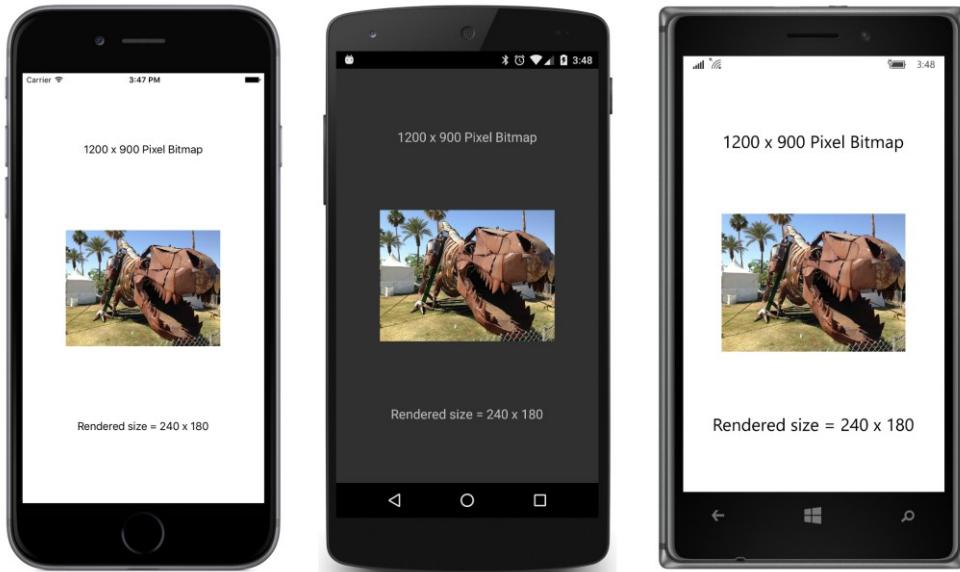
    HorizontalOptions = "Centrar"
    SizeChanged = "OnImageSizeChanged" />
</Imagen>

<Etiqueta x: Nombre = "etiqueta"
    Tamaño de fuente = "Medio"
    VerticalOptions = "CenterAndExpand"
    HorizontalOptions = "Centrar" />

</StackLayout>
</Pagina de contenido>

```

Si el análisis anterior acerca de dimensionamiento es correcta y todo va bien, este mapa de bits debe mirar aproximadamente el mismo tamaño en las tres plataformas relativas a la anchura de la pantalla, así como proporcionar una resolución más alta fidelidad que el programa anterior:



Con este conocimiento sobre dimensionamiento de mapas de bits, ahora es posible hacer que un lector poco de libros electrónicos con imágenes, porque lo que es el uso de un libro sin imágenes?

Este lector de libros electrónicos muestra un desplazable StackLayout con el texto completo del Capítulo 7 de Lewis Carroll *Las aventuras de Alicia en el País de las Maravillas*, incluyendo tres de las ilustraciones originales de John Tenniel. El texto y las ilustraciones fueron descargados de la página web de la Universidad de Adelaida. Las ilustraciones se incluyen como recursos incrustados en el **MadTeaParty** proyecto. Tienen los mismos nombres y tamaños como los de la página web. Los nombres se refieren a los números de página en el libro original:

- image113.jpg - 709 × 553
- image122.jpg - 485 × 545
- image129.jpg - 670 × 596

Recordemos que el uso de `WidthRequest` para `Imagen` elementos en una `StackLayout` sólo puede reducir el tamaño de los mapas de bits prestados. Estos mapas de bits no son lo suficientemente amplia como para asegurar que todos ellos se reducen a un tamaño adecuado en las tres plataformas, pero es interesante examinar los resultados de todos modos porque esto es mucho más cerca de un ejemplo real.

los `MadTeaParty` programa utiliza un estilo implícito para `Imagen` para establecer el `WidthRequest` propiedad a un valor correspondiente a 1,5 pulgadas. Al igual que en el ejemplo anterior, este valor es 240.

Para los tres dispositivos utilizados para estas capturas de pantalla, esta anchura corresponde a:

- 480 píxeles en el iPhone 6
- 720 píxeles en el Nexus 5 Android
- 540 píxeles en el Nokia Lumia 925 funcionando 10 en Windows Mobile

Esto significa que las tres imágenes se reducirán en tamaño en el iPhone 6, y todos ellos tendrán un ancho representado de 240 unidades independientes del dispositivo.

Sin embargo, ninguna de las tres imágenes se reducirá en tamaño en el Nexus 5 porque todos tienen estrechos anchos de píxeles que el número de píxeles de 1,5 pulgadas. Las tres imágenes tendrán un ancho representado de (respectivamente) 236, 162, y 223 unidades independientes del dispositivo en el Nexus 5. (que es el ancho de pixel dividido por 3.)

En el dispositivo móvil de Windows 10, dos se encogen y la otra no.

Vamos a ver si las predicciones son correctas. El archivo XAML incluye una Color de fondo establecer en el elemento raíz de que los colores de toda la página en blanco, como corresponde a un libro. los Estilo definiciones se limitan a una recursos en el diccionario `StackLayout`. Un estilo para el título del libro se basa en el dispositivo `TitleStyle` pero con el texto negro y centrado, y dos estilos implícitos para `Etiqueta` y `Imagen` serviría para el estilo de la mayoría de las `Etiqueta` elementos y los tres `Imagen` elementos. Sólo el primer y último párrafos del texto del capítulo se muestran en este listado del archivo XAML:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    xmlns: sys = "clr-espacio de nombres: System; montaje = mscorelib "
    xmlns: locales = "CLR-espacio de nombres: MadTeaParty "
    x: Class = " MadTeaParty.MadTeaPartyPage "
    Color de fondo = " Blanco " >

    < ContentPage.Padding >
        < OnPlatform x: TypeArguments = " Espesor "
            iOS = " 5, 20, 5, 0 "
            Androide = " 5, 0 "
            WinPhone = " 5, 0 " />
    </ ContentPage.Padding >

    < ScrollView >
        < StackLayout Espaciado = " 10 " >
            < StackLayout.Resources >
```

```

< ResourceDictionary >
    < Estilo x: Key = "TitleLabel" 
        Tipo de objetivo = " Etiqueta "
        BaseResourceKey = " TextStyle "
        < Setter Propiedad = " Color de texto " Valor = " Negro " />
        < Setter Propiedad = " HorizontalTextAlignment " Valor = " Centrar " />
    </ Estilo >

    <! - estilos implícitos ->
    < Estilo Tipo de objetivo = " Etiqueta "
        BaseResourceKey = " Tipo de cuerpo "
        < Setter Propiedad = " Color de texto " Valor = " Negro " />
    </ Estilo >

    < Estilo Tipo de objetivo = " Imagen " 
        < Setter Propiedad = " WidthRequest " Valor = " 240 " />
    </ Estilo >

    <! - Cuarto guión pulgadas para la poesía ->
    < Espesor x: Key = " poemIndent " > 40, 0, 0, 0 </ Espesor >
</ ResourceDictionary >
</ StackLayout.Resources >

<! - Texto e imágenes de http://ebooks.adelaide.edu.au/c/carroll/lewis/alice/ ->
< StackLayout Espaciado = " 0 " >
    < Etiqueta Texto = " Las aventuras de Alicia en el País de las Maravillas "
        Estilo = " { } DynamicResource TitleLabel "
        FontAttributes = " Itálico " />

    < Etiqueta Texto = " de Lewis Carroll "
        Estilo = " { } DynamicResource TitleLabel " />
</ StackLayout >

< Etiqueta Estilo = " { } DynamicResource SubtitleStyle "
    Color de texto = " Negro "
    HorizontalTextAlignment = " Centrar " >
    < Label.FormattedText >
        < FormattedString >
            < Largo Texto = " capítulo VII " />
            < Largo Texto = " {X: sys:estáticas: Environment.NewLine} " />
            < Largo Texto = " Una merienda de locos " />
        </ FormattedString >
    </ Label.FormattedText >
</ Etiqueta >

< Etiqueta Texto =
    " Había un cuadro que figura debajo de un árbol frente a la
    casa, y la Liebre de Marzo y el Sombrerero estaban tomando el té en
    : un lirón estaba sentado entre ellos, profundamente dormido, y
    los otros dos fueron de usarlo como un cojín, descansando su
    codos en él, y hablando sobre su cabeza. 'Muy incómodo
    para el Lirón, pensó Alicia; 'Solamente, ya que es dormido,
    supongo que no le importa.' " />
    ...

```

```
...
...
< Etiqueta >
< Label.FormattedText >
< FormattedString >
< Lapso Texto =
" Una vez más se encontró en el largo pasillo, y cerca de
la mesita de cristal. 'Ahora, voy a manejar mejor esta vez,'
se dijo, y comenzó tomando la pequeña de oro
clave, y desbloquear la puerta que daba al jardín. Entonces
se fue a trabajar a mordisquear el hongo (que había mantenido una
pedazo de ella en el bolílico) hasta que ella estaba a punto de un pie de altura:
a continuación, mientras caminaba por el pequeño pasaje: y " />
< Lapso Texto = " entonces " FontAttributes = " Ítálico " />
< Lapso Texto =
"- se encontró por fin en el hermoso jardín,
entre las flores brillantes camas y las frescas fuentes. " />
< / FormattedString >
</ Label.FormattedText >
</ Etiqueta >
</ StackLayout >
</ ScrollView >
</ Pagina de contenido >
```

El tres Imagen elementos simplemente hacen referencia a los tres recursos incrustados y se les da un ajuste de la WidthRequest la propiedad a través del estilo implícita:

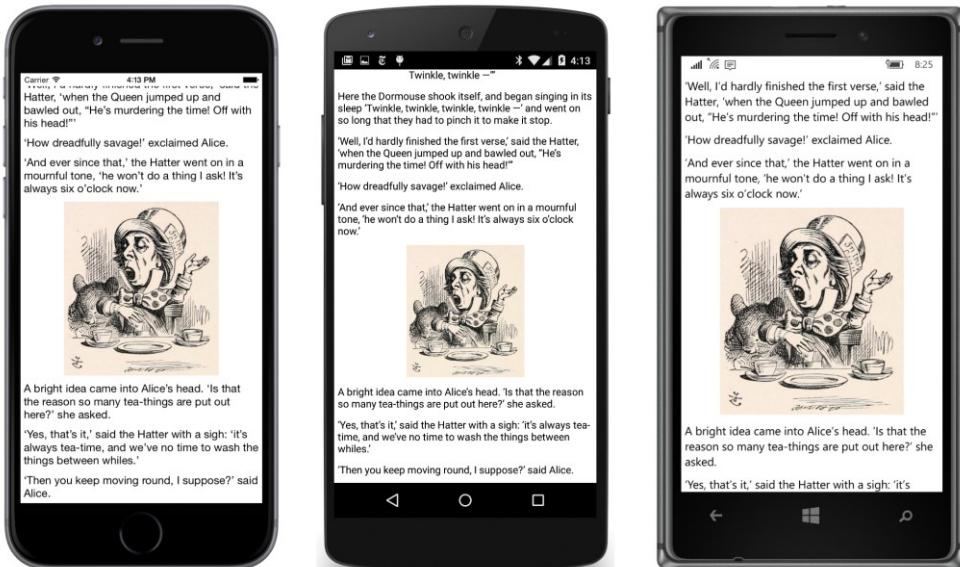
```
< Imagen Fuente = " {Locales: MadTeaParty.Images.image113.jpg ImageResource} " />
...
< Imagen Fuente = " {Locales: MadTeaParty.Images.image122.jpg ImageResource} " />
...
< Imagen Fuente = " {Locales: MadTeaParty.Images.image129.jpg ImageResource} " />
```

Aquí está la primera imagen:



Es bastante constante entre las tres plataformas, a pesar de que se muestre en su anchura natural de 709 píxeles en el Nexus 5, pero que está muy cerca de los 720 píxeles que una anchura de 240 unidades independientes del dispositivo implica.

La diferencia es mucho mayor con la segunda imagen:



Esto se muestra en su tamaño de píxel en el Nexus 5, que corresponde a 162 unidades independientes del dispositivo, pero se muestra con una anchura de 240 unidades en el iPhone 6 y el Nokia Lumia 925.

Aunque las fotos no se ven mal en cualquiera de las plataformas, para que hagan todo sobre el mismo tamaño requeriría comenzando con mapas de bits grandes.

La navegación y la espera

Otra característica de Imagen se demuestra en el **ImageBrowser** programa, que te permite navegar por las fotografías de archivo utilizados para algunos de los ejemplos de este libro. Como se puede ver en el siguiente archivo XAML, una **Imagen** elemento comparte la pantalla con una Etiqueta y dos Botón puntos de vista. Tenga en cuenta que una Propiedad-cambiado gestora se establece en el **Imagen**. Que ha aprendido en el capítulo 11, "La infraestructura enlazable," que la **PropertyChanged** controlador es implementado por **BindableObject** y se dispara siempre que una propiedad enlazable cambia de valor.

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns:x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " ImageBrowser.ImageBrowserPage " >

    < ContentPage.Padding >
        < OnPlatform x: TypeArguments = " Espesor "
            iOS = " 0, 20, 0, 0 " />
    </ ContentPage.Padding >

    < StackLayout >
        < Imagen x: Nombre = " imagen "
            VerticalOptions = " CenterAndExpand "
            PropertyChanged = " OnImagePropertyChanged " />

        < Etiqueta x: Nombre = " filenameLabel "
            HorizontalOptions = " Centrar " />

        < ActivityIndicator x: Nombre = " activityIndicator " />

        < StackLayout Orientación = " Horizontal " >
            < Botón x: Nombre = " prevButton "
                Texto = " Anterior "
                Está habilitado = " falso "
                HorizontalOptions = " CenterAndExpand "
                hecho clic = " OnPreviousButtonClicked " />

            < Botón x: Nombre = " siguiente botón "
                Texto = " Siguiente "
                Está habilitado = " falso "
                HorizontalOptions = " CenterAndExpand "
                hecho clic = " OnNextButtonClicked " />
        </ StackLayout >
    </ StackLayout >
</ Página de contenido >
```

También en esta página es una **ActivityIndicator**. Por lo general, utiliza este elemento cuando un programa está a la espera para una operación larga para completar (como la descarga de un mapa de bits), pero no puede proporcionar ninguna información sobre el progreso de la operación. Si su programa sabe qué fracción de la operación se ha completado, se puede utilizar una Barra de progreso en lugar. (Barra de progreso se demuestra en el siguiente capítulo).

los ActivityIndicator tiene una propiedad booleana llamada Esta corriendo. Normalmente, esa propiedad es falso y el ActivityIndicator es invisible. Establecer la propiedad en cierto para hacer que el ActivityIndicator visible. Las tres plataformas implementar una animación visual para indicar que el programa está funcionando, pero se ve un poco diferente en cada plataforma. En iOS es una rueda que gira, y en Android es un círculo parcial de giro. En los dispositivos de Windows, una serie de puntos mueve por la pantalla.

Para proporcionar acceso a navegar por las imágenes, la ImageBrowser tiene que descargar un archivo JSON con una lista de todos los nombres de archivo. Con los años, varias versiones de .NET han introducido varias clases de objetos capaces de descargar a través de Internet. Sin embargo, no todos estos están disponibles en la versión de .NET que está disponible en una biblioteca de clases portátil que tiene el perfil compatible con Xamarin.Forms. Una clase que está disponible es WebRequest y su clase descendiente HttpWebRequest.

los WebRequest.Create método devuelve una WebRequest método basado en un URI. (El valor de retorno es en realidad una HttpWebRequest objeto.) La BeginGetResponse método requiere una función de devolución de llamada que se llama cuando el Corriente haciendo referencia a la URI está disponible para su acceso. los Corriente es accesible desde una llamada a EndGetResponse y GetResponseStream.

Una vez que el programa obtiene acceso a la Corriente objeto en el código siguiente, se utiliza el DataContractJsonSerializer clase juntos con las funciones integradas ImageList clase definida cerca de la parte superior de la ImageBrowserPage clase para convertir el archivo a un JSON ImageList objeto:

```
p\xedblico clase parcial ImageBrowserPage : Pagina de contenido
{
    [DataContract]
    clase ImageList
    {
        [DataMember (Nombre = "Fotos")]
        p\xedblico Lista < cuadra > Fotos = nulo ;
    }

    WebRequest solicitud;
    ImageList imageList;
    Ent imageListIndex = 0;

    p\xedblico ImageBrowserPage ()
    {
        InitializeComponent ();

        // lista de fotografías de archivo Obtener.
        Uri uri = nuevo Uri ("Https://developer.xamarin.com/demo/stock.json");
        request = WebRequest.Create (URI);
        request.BeginGetResponse (WebRequestCallback, nulo );
    }

    vac\xedo WebRequestCallback ( IAsyncResult resultado )
    {
        Dispositivo .BeginInvokeOnMainThread (() =>
        {
            tratar
        });
    }
}
```

```
Corriente flujo = request.EndGetResponse (resultado) .GetResponseStream ();  
  
// deserializar el JSON en imageList;  
var jsonSerializer = nuevoDataContractJsonSerializer ( tipo de ( ImageList ));  
imageList = ( ImageList ) jsonSerializer.ReadObject (corriente);  
  
Si (ImageList.Photos.Count > 0)  
    FetchPhoto ();  
}  
captura ( Excepción Exc )  
{  
    filenameLabel.Text = exc.Message;  
}  
});  
}  
  
vacío OnPreviousButtonClicked ( objeto remitente, EventArgs args )  
{  
    imageListIndex--;  
    FetchPhoto ();  
}  
  
vacío OnNextButtonClicked ( objeto remitente, EventArgs args )  
{  
    imageListIndex++;  
    FetchPhoto ();  
}  
  
vacío FetchPhoto ()  
{  
    // Preparar para la nueva imagen.  
    image.Source = nulo ;  
    cuerda url = imageList.Photos [imageListIndex];  
  
    // Establecer el nombre del archivo.  
    filenameLabel.Text = url.Substring (url.LastIndexOf ('?') + 1);  
  
    // Crear el UriImageSource.  
    UriImageSource ImageSource = nuevo UriImageSource  
    {  
        = uri nuevo Uri (Uri + "? Ancho = 1080"),  
        CacheValidity = Espacio de tiempo .FromDays (30)  
    };  
  
    // Establecer la Fuente de la imagen.  
    image.Source = ImageSource;  
  
    // Activar o desactivar los botones.  
    prevButton.IsEnabled = imageListIndex > 0;  
    nextButton.IsEnabled = imageListIndex < imageList.Photos.Count - 1;  
}  
  
vacío OnImagePropertyChanged ( objeto remitente, PropertyChangedEventArgs args )  
{
```

```
Si (Args.PropertyName == "Esta cargando")
{
    activityIndicator.IsRunning = ((Imagen) Remitente).IsLoading;
}
}
```

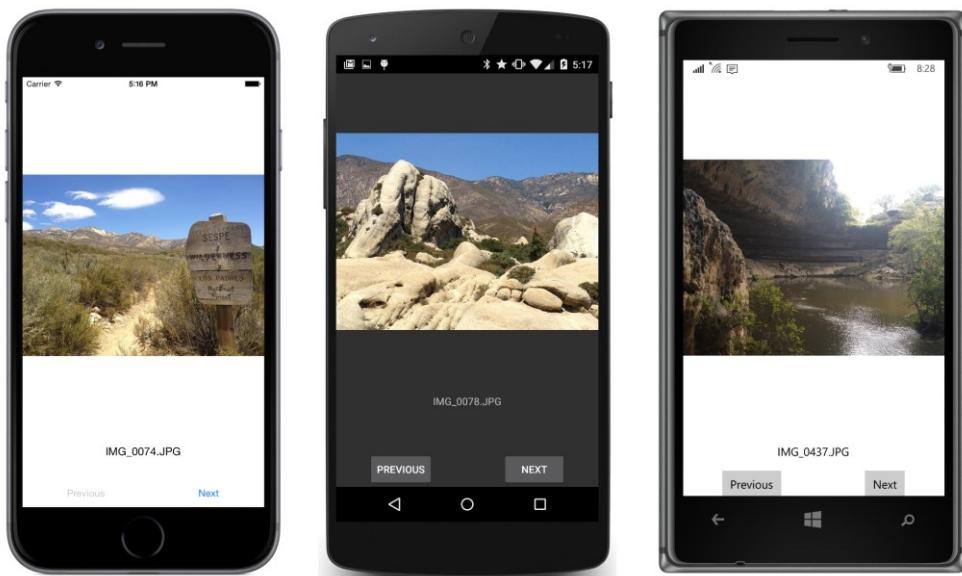
Todo el cuerpo de la WebRequestCallback método está encerrado en una función lambda que es el argumento de la Device.BeginInvokeOnMainThread método. WebRequest descarga el archivo referenciado por el URI en un subproceso secundario de la ejecución. Esto asegura que la operación no bloquea hilo principal del programa, que está manejando la interfaz de usuario. El método de devolución de llamada también ejecuta en este hilo secundario. Sin embargo, los objetos de interfaz de usuario en una aplicación Xamarin.Forms sólo se puede acceder desde el hilo principal.

El propósito de Device.BeginInvokeOnMainThread método consiste en solucionar este problema. El argumento de este método se pone en cola para funcionar en el hilo principal del programa y se puede acceder con seguridad a los objetos de interfaz de usuario.

Al hacer clic en los dos botones, las llamadas a FetchPhoto utilizar UriImageSource para descargar un nuevo mapa de bits. Esto puede tardar un segundo o así. Los Imagen clase define una propiedad booleana llamada Esta cargando es decir cierto cuando Imagen está en el proceso de carga (o descarga) un mapa de bits. Esta cargando está respaldado por la propiedad enlazable IsLoadingProperty. Eso también significa que cada vez Esta cargando cambia de valor, una PropertyChanged evento se dispara. El programa utiliza el PropertyChanged handler- caso del OnImagePropertyChanged método en la parte inferior de la clase para ajustar el Esta corriendo propiedad de la ActivityIndicator al mismo valor que la Esta cargando propiedad de Imagen.

Usted verá en el capítulo 16, "El enlace de datos," cómo sus aplicaciones se pueden vincular propiedades como Esta cargando y Esta corriendo para que mantengan el mismo valor sin ningún tipo de controladores de eventos explícitos.

A continuación se ImageBrowser en acción:



Algunas de las imágenes tienen definido el indicador de orientación EXIF, y si la plataforma en particular ignora esa bandera, la imagen se muestra de lado.

Si ejecuta este programa en modo horizontal, descubrirá que los botones desaparecen. Una opción de diseño mejor para este programa es una Cuadricula, lo que se demuestra en el capítulo 17.

mapas de bits de streaming

Si el Fuente de imagen clase no tenía FromUri o FromResource métodos, aún podrán acceder a mapas de bits a través de Internet o almacenados como recursos en el PCL. Puede hacer ambas empleos, así como varios otros, con ImageSource.FromStream o el StreamImageSource clase.

los ImageSource.FromStream método es algo más fácil de usar que las StreamImageSource, pero ambos son un poco extraño. El argumento para ImageSource.FromStream no es un Corriente objeto, sino una Func objeto (un método sin argumentos) que devuelve una Corriente objeto. los Corriente propiedad de CorrienteFuente de imagen no es igualmente una Corriente objeto, sino una Func objeto que tiene una CancellationToken argumento y devuelve una Tarea <ruta> objeto.

Acceder a los flujos

los BitmapStreams programa contiene un archivo XAML con dos Imagen elementos de espera de mapas de bits, cada uno de los cuales se encuentra en el archivo de código subyacente mediante el uso de ImageSource.FromStream:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
      xmlns:x = " http://schemas.microsoft.com/winfx/2009/xaml "
```

```

x: Class = "BitmapStreams.BitmapStreamsPage" >
< StackLayout >
    < Imagen x: Nombre = "Imagen1" 
        HorizontalOptions = "Centrar" 
        VerticalOptions = "CenterAndExpand" />

    < Imagen x: Nombre = "Imagen2" 
        HorizontalOptions = "Centrar" 
        VerticalOptions = "CenterAndExpand" />
</ StackLayout >
</ Pagina de contenido >

```

El primero Imagen se establece desde un recurso incrustado en el PCL; la segunda se establece a partir de un mapa de bits de acceso a través de la web.

En el **Gato negro** programa en el Capítulo 4, “Desplazamiento de la pila”, que vio cómo obtener una Corriente objeto para cualquier recurso almacenado con una **construir Acción de EmbeddedResource** en el PCL. Se puede utilizar esta misma técnica para acceder a un mapa de bits almacenado como un recurso incrustado:

```

pública clase parcial BitmapStreamsPage : Pagina de contenido
{
    pública BitmapStreamsPage ()
    {
        InitializeComponent ();

        // Cargar incrustado de mapa de bits de recursos.
        cuerda de resourceId = "BitmapStreams.Images.IMG_0722_512.jpg";
        image1.Source = Fuente de imagen .FromStream () =>
        {
            Montaje montaje = GetType () GetTypeInfo () Asamblea..;
            Corriente flujo = assembly.GetManifestResourceStream (de resourceId);
            regreso corriente;
        });
        ...
    }
}

```

El argumento para `ImageSource.FromStream` se define como una función que devuelve una Corriente objeto, de modo que el argumento que aquí se expresa como una función lambda. La llamada a la `GetType` método devuelve el tipo de la `BitmapStreamsPage` clase, y `GetTypeInfo` proporciona más información acerca de ese tipo, incluyendo el `Montaje` objeto que contiene el tipo. Eso es `BitmapStream` conjunto de PCL, que es el conjunto con el recurso incrustado. `GetManifestResourceStream` devuelve una Corriente objeto, que es el valor de retorno que `ImageSource.FromStream` quiere.

Si alguna vez necesita un poco de ayuda con los nombres de estos recursos, el `GetManifestResourceNames` devuelve un array de objetos de cadena con todos los ID de los recursos en el PCL. Si no puede averiguar por qué su `GetManifestResourceStream` no está funcionando, compruebe en primer lugar asegurarse de que sus recursos tienen una **construir Acción de EmbeddedResource**, y luego llamar `GetManifestResourceNames` para obtener todos los ID de los recursos.

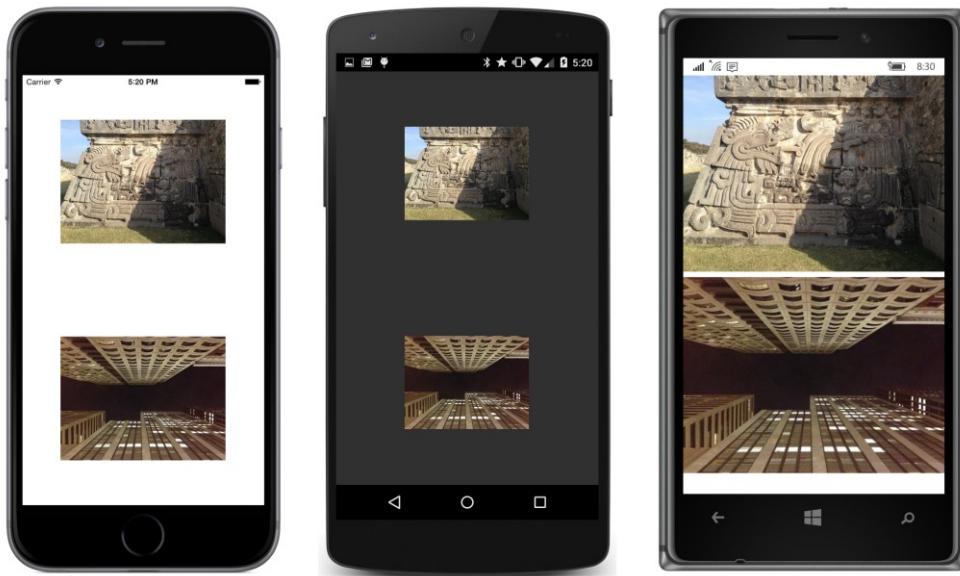
Para descargar un mapa de bits a través de Internet, puede utilizar el mismo WebRequest método demostró anteriormente en la **ImageBrowser** programa. En este programa, el BeginGetResponse devolución de llamada es una función lambda:

```
pública clase parcial BitmapStreamsPage : Pagina de contenido
{
    pública BitmapStreamsPage ()
    {
        ...
        // mapa de bits de carga web.
        Uri uri = nuevo Uri ("Https://developer.xamarin.com/demo/IMG_0925.JPG?width=512");
        WebRequest request = WebRequest .Create (URI);
        request.BeginGetResponse (( IAsyncResult arg) =>
        {
            Corriente flujo = request.EndGetResponse (arg) .GetResponseStream ();

            Si (Device.OS == TargetPlatform.WinPhone ||
                Device.OS == TargetPlatform.Windows)
            {
                MemoryStream memStream = nuevo MemoryStream ();
                stream.CopyTo (memStream);
                memStream.Seek (0, SeekOrigin .Empezar);
                flujo = memStream;
            }
            Fuente de imagen ImageSource = Fuente de imagen .FromStream ( () => corriente);
            Dispositivo .BeginInvokeOnMainThread ( () => image2.Source = ImageSource);
        }, nulo );
    }
}
```

los BeginGetResponse devolución de llamada también contiene otras dos funciones lambda incrustados! La primera línea de la devolución de llamada obtiene la Corriente objeto para el mapa de bits. Esta Corriente objeto no es muy adecuado para Windows en tiempo de ejecución por lo que los contenidos se copian en una MemoryStream.

La siguiente instrucción utiliza una función lambda corto como el argumento de ImageSource.FromStream para definir una función que devuelve esa corriente. La última línea de la BeginGetResponse devolución de llamada es una llamada a Device.BeginInvokeOnMainThread para establecer el Fuente de imagen oponerse a la Fuente propiedad de la Imagen.



Podría parecer como si usted tiene más control sobre la descarga de imágenes mediante el uso de **WebRequest** y **ImageSource.FromStream** que con **ImageSource.FromUri**, pero el **ImageSource.FromUri** método tiene una gran ventaja: se almacena en caché los mapas de bits descargados en un área de almacenamiento privado a la aplicación. Como hemos visto, se puede apagar el almacenamiento en caché, pero si usted está utilizando **ImageSource.FromStream** en lugar de **ImageSource.FromUri**, puede encontrarse con la necesidad de almacenar en caché las imágenes, y que sería un trabajo mucho más grande.

Generación de mapas de bits en tiempo de ejecución

Las tres plataformas soportan el formato de archivo BMP, que se remonta a los inicios de Microsoft Windows. A pesar de su antigua herencia, el formato de archivo BMP es ahora bastante estandarizada con más extensa información del encabezado.

Aunque hay algunas opciones BMP que permiten un poco de compresión rudimentaria, la mayoría de los archivos BMP no están comprimidos. Esta falta de compresión suele ser considerado como una desventaja del formato de archivo BMP, pero en algunos casos no es una desventaja en absoluto. Por ejemplo, si desea generar un mapa de bits en tiempo de ejecución algorítmica, es *mucho* más fácil de generar un mapa de bits sin comprimir en lugar de uno de los formatos de archivos comprimidos. (De hecho, incluso si has tenido una función de biblioteca para crear un archivo JPEG o PNG, que le aplique esa función a los datos de píxeles sin compresión).

Puede crear un mapa de bits en tiempo de ejecución algorítmica llenando una **MemoryStream** con las cabeceras de los archivos BMP y los datos de píxeles y luego pasa que **MemoryStream** al **ImageSource.FromStream** método. los **BmpMaker** clase en el **Xamarin.FormsBook.Toolkit** biblioteca demuestra. Se crea una BMP en la memoria utilizando un 32-bit pixel formato de 8 bits cada uno para el rojo, verde, azul y alfa (opacidad) chan-

Nels. los **BmpMaker** clase se codifica con el rendimiento en mente, con la esperanza de que podría ser utilizado para la animación. Tal vez algún día será, pero en este capítulo la única manifestación es un simple degradado de color.

El constructor crea una byte matriz denominada buffer que almacena todo el archivo BMP a partir de la información de cabecera y seguidos por los bits de pixeles. El constructor utiliza luego una MemoryStream para escribir la información de cabecera al comienzo de esta memoria:

```

clase pública BmpMaker
{
    const int headerSize = 54;
    byte de sólo lectura [] buffer;

    público BmpMaker (En t anchura, En t altura)
    {
        Anchura = ancho;
        Altura = altura;

        En t numPixels = Anchura * Altura;
        En t numPixelBytes = 4 * numPixels;
        En t FileSize = headerSize + numPixelBytes;
        buffer = nuevo byte [tamaño del archivo];

        // cabeceras de escritura en MemoryStream y por lo tanto, la memoria intermedia.
        utilizando (MemoryStream MemoryStream = nuevo MemoryStream (buffer))
        {
            utilizando (BinaryWriter escritor = nuevo BinaryWriter (MemoryStream, codificación .utf8))
            {
                // Construir cabecera BMP (14 bytes).
                escritor.Write (nuevo car [] { 'SEGUNDO', 'METRO' }); // Firma
                escritor.Write (fileSize); // Tamaño del archivo
                escritor.Write ((corto) 0); // Reservados
                escritor.Write ((corto) 0); // Reservados
                escritor.Write (headerSize); // Offset a pixeles

                // Construir BITMAPINFOHEADER (40 bytes).
                escritor.Write (40); // tamaño de la cabecera
                escritor.Write (anchura); // anchura Pixel
                escritor.Write (altura); // altura de píxeles
                escritor.Write ((corto) 1); // Planes
                escritor.Write ((corto) 32); // bits por pixel
                escritor.Write (0); // compresión
                escritor.Write (numPixelBytes); // Tamaño de imagen en bytes
                escritor.Write (0); // X pixeles por metro
                escritor.Write (0); // Y pixeles por metro
                escritor.Write (0); // Número colores en la tabla de colores
                escritor.Write (0); // Importante recuento de color
            }
        }
    }

    public int Anchura
    {

```

```

conjunto privado ;
obtener ;
}

public int Altura
{
    conjunto privado ;
    obtener ;
}

public void SetPixel ( Ent fila, Ent columna, Color color)
{
    SetPixel (fila, col, ( Ent ) (255 * color.R),
              ( Ent ) (255 * color.G),
              ( Ent ) (255 * color.B),
              ( Ent ) (255 * color.A));
}

public void SetPixel ( Ent fila, Ent columna, Ent r, Ent g, Ent b, Ent a = 255)
{
    Ent index = (fila * Ancho + col) * 4 + headerSize;
    buffer [índice + 0] = ( byte )segundo;
    buffer [índice + 1] = ( byte )gramo;
    buffer [índice + 2] = ( byte ) R;
    buffer [índice + 3] = ( byte )un;
}

público Fuente de imagen Generar()
{
    // Crear MemoryStream de búfer con mapa de bits.
    MemoryStream MemoryStream = nuevo MemoryStream (buffer);

    // Convertir a StreamImageSource.
    Fuente de imagen ImageSource = Fuente de imagen .FromStream () =>
    {
        regreso MemoryStream;
    });
    regreso fuente de imagen;
}
}

```

Después de crear una `BmpMaker` objeto, un programa puede llamar a uno de los dos `SetPixel` métodos para establecer un color en una fila y columna particular. Al hacer muchas llamadas, las `SetPixel` llamada que utiliza un `Color` valor es significativamente más lento que el que acepta roja explícita, verde, azul y los valores.

El último paso es llamar al `Generar` método. Este método crea la instancia de otro `MemoryStream` objeto basado en el buffer array y lo utiliza para crear una `FileImageSource` objeto. Puedes llamar gene-comió múltiples veces antes de aceptar los nuevos datos de píxeles. El método crea un nuevo `MemoryStream` porque cada vez `ImageSource.FromStream` cierra la Corriente objeto cuando se termine con ella.

los `DiyGradientBitmap` programa- "hágalo usted mismo" es sinónimo de "hágalo usted mismo" -demostraba cómo utilizar

BmpMaker para hacer un mapa de bits con un gradiente simple y mostrarlo para llenar la página. El archivo XAML incluye la Imagen elemento:

```
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " DiyGradientBitmap.DiyGradientBitmapPage " >

    < ContentPage.Padding >
        < OnPlatform x: TypeArguments = " Espesor "
            iOS = " 0, 20, 0, 0 " />
    </ ContentPage.Padding >

    < Imagen x: Nombre = " imagen "
        Aspecto = " Llenar " />

</ Pagina de contenido >
```

El archivo de código subyacente instancia un BmpMaker y bucles a través de las filas y columnas del mapa de bits para crear un gradiente que va desde rojo en la parte superior a azul en la parte inferior:

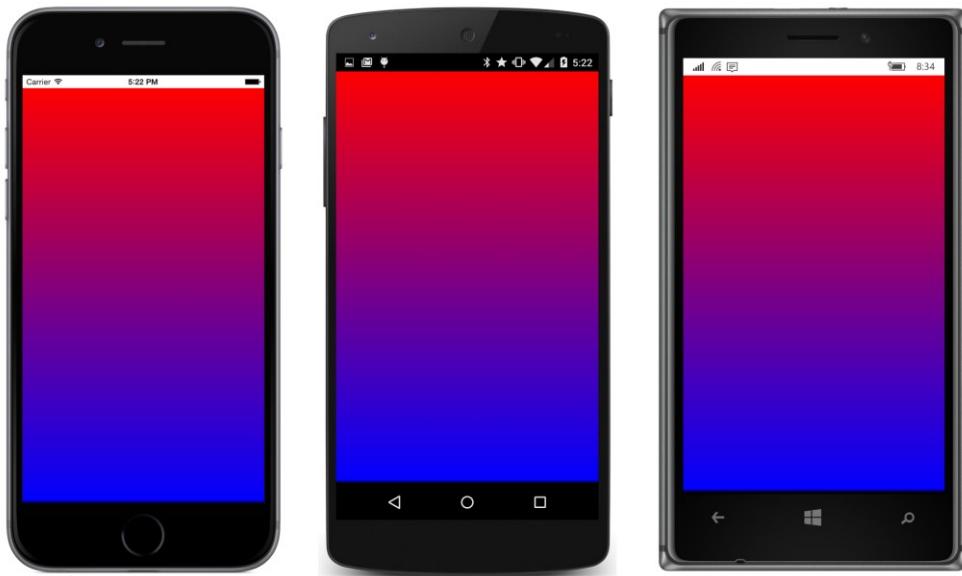
```
público clase parcial DiyGradientBitmapPage : Pagina de contenido
{
    público DiyGradientBitmapPage ()
    {
        InitializeComponent ();

        En t filas = 128;
        En t cols = 64;
        BmpMaker bmpMaker = nuevo BmpMaker (cols, filas);

        para ( En t fila = 0; fila <filas; fila++)
            para ( En t col = 0; col <cols; col++)
            {
                bmpMaker.SetPixel (fila, col, * la fila 2, 0, 2 ^ (128 - fila));
            }

        Fuente de imagen ImageSource = bmpMaker.Generate ();
        image.Source = ImageSource;
    }
}
```

Aquí está el resultado:



Ahora usa tu imaginación y ver lo que puede hacer con BmpMaker.

mapas de bits específicos de la plataforma

Como hemos visto, puede cargar mapas de bits a través de Internet o desde el proyecto PCL compartido. También puede cargar mapas de bits almacenados como recursos en los proyectos de plataforma individuales. Las herramientas para este trabajo son los `ImageSource.FromFile` método estático y el correspondiente `FileImageSource` clase.

Probablemente utilice este servicio sobre todo para los mapas de bits conectadas con los elementos de interfaz de usuario. los icono bienes en Opción del menú y `ToolbarItem` es de tipo `FileImageSource`. los Imagen bienes en Botón es también de tipo `FileImageSource`.

otros dos usos de `FileImageSource` no será discutido en este capítulo: la Página clase define una `Icono` propiedad de tipo `FileImageSource` y una `Imagen de fondo` propiedad de tipo `cuerda`, pero que se supone que es el nombre de un mapa de bits almacenado en el proyecto de plataforma.

El almacenamiento de mapas de bits en los proyectos de plataforma individuales permite un alto nivel de especificidad plataforma. Se podría pensar que se puede obtener el mismo grado de especificidad plataforma mediante el almacenamiento de mapas de bits para cada plataforma en el proyecto PCL y el uso de la `Device.OnPlatform` método o la `OnPlatform` clase para seleccionarlos. Sin embargo, como pronto descubrirá, las tres plataformas tienen disposiciones para almacenar mapas de bits de diferentes píxeles de resolución y luego acceder automáticamente el óptimo. Usted puede tomar ventaja de esta característica valiosa sólo si las mismas plataformas individuales se cargan los mapas de bits, y este es el caso, sólo cuando se utiliza `ImageSource.FromFile` y `FileImageSource`.

Los proyectos de plataforma en una solución Xamarin.Forms recién creado ya contienen varios mapas de bits. En el proyecto de iOS, encontrarás estos en el **recursos carpeta**. En el proyecto Android, que están en las subcarpetas de la **recursos carpeta**. En las ventanas de distintos proyectos, que están en el **Bienes carpetas y subcarpetas**. Estos mapas de bits son iconos de aplicaciones y pantallas de bienvenida, y te quieren reemplazarlos cuando se prepara para traer una aplicación al mercado.

Vamos a escribir un pequeño proyecto llamado **PlatformBitmaps** con acceso a un ícono de la aplicación de cada proyecto de plataforma y muestra el tamaño de la rendido **Imagen** elemento. Si está utilizando **FileImageSource** para cargar el mapa de bits (ya que este programa lo hace), es necesario establecer el **Archivo** propiedad a un cuerdas con el nombre de archivo del mapa de bits. Casi siempre, que va a utilizar **Device.OnPlatform** en clave o **OnPlatform** en XAML para especificar los tres nombres de archivo:

```
clase pública PlatformBitmapsPage : Pagina de contenido
{
    público PlatformBitmapsPage ()
    {
        Imagen image = nuevo Imagen
        {
            fuente = nuevo FileImageSource
            {
                file = Dispositivo .OnPlatform (IOS: "Icono-Pequeño-40.png",
                                                Androide: "Icon.png",
                                                WinPhone: "Activos / StoreLogo.png" )
            },
            HorizontalOptions = LayoutOptions .Centrar,
            VerticalOptions = LayoutOptions .CenterAndExpand
        };
    }

    Etiqueta etiqueta = nuevo Etiqueta
    {
        Tamaño de Letra = Dispositivo .GetNamedSize ( NamedSize .Medio, tipo de ( Etiqueta )),
        HorizontalOptions = LayoutOptions .Centrar,
        VerticalOptions = LayoutOptions .CenterAndExpand
    };

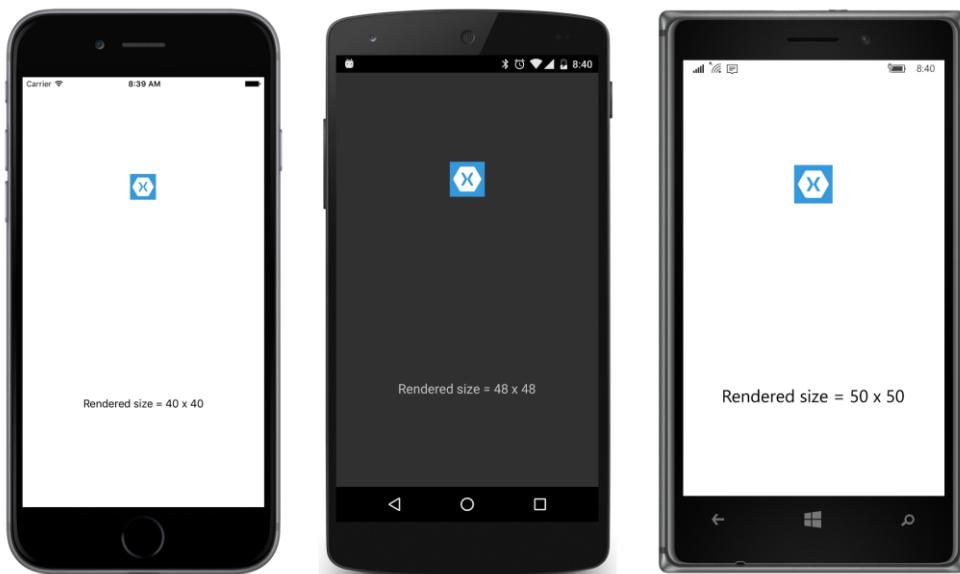
    image.SizeChanged += (remitente, args) =>
    {
        label.text = Cuerda .Formato( "Tamaño de rendida = {0} x {1}" ,
                                       image.Width, image.Height);
    };

    content = nuevo StackLayout
    {
        Los niños =
        {
            imagen,
            etiqueta
        }
    };
}
```

Al acceder a un mapa de bits almacenado en el recursos carpeta del proyecto o el iOS recursos

carpeta (o subcarpetas) del proyecto Android, no prefacio, el nombre de archivo con un nombre de carpeta. Estas carpetas son los repositorios estándar para mapas de bits en estas plataformas. Pero los mapas de bits pueden estar en cualquier parte en el proyecto de Windows Phone o Windows (incluyendo la raíz del proyecto), por lo que se requiere que el nombre de la carpeta (si lo hay).

En los tres casos, el icono por defecto es el famoso logotipo de Xamarin hexagonal (cariñosamente conocido como el Xamagon), pero cada plataforma tiene diferentes convenciones para su tamaño de los iconos, por lo que los tamaños son diferentes prestados:



Si comienza a explorar los mapas de bits del icono en los proyectos de iOS y Android, que podría ser un poco confuso: parece que hay varios mapas de bits con el mismo nombre (o nombres similares) en los proyectos de iOS y Android.

Es el momento de profundizar más en el tema de la resolución de mapa de bits.

resoluciones de mapa de bits

El nombre del archivo de mapa de bits especificado en iOS **PlatformBitmaps** Icono es pequeña o 40.png, pero si nos fijamos en el recursos carpeta del proyecto IOS, verá tres archivos con variaciones de ese nombre. Todos ellos tienen diferentes tamaños:

- Icono-Pequeño-40.png - 40 píxeles cuadrados
- Icon-Small-40@2x.png - 80 píxeles cuadrados
- Icon-Small-40@3x.png - 120 píxeles cuadrados

Como usted descubrió al principio de este capítulo, cuando una Imagen es un hijo de una StackLayout, iOS muestra el mapa de bits en su tamaño de pixel con un mapeo uno a uno entre los píxeles del mapa de bits y los píxeles de la pantalla. Esta es la visualización óptima de un mapa de bits.

Sin embargo, en el iPhone 6 simulador utilizado en la captura de pantalla, la Imagen tiene un tamaño rindió de 40 unidades independientes del dispositivo. En el iPhone 6 hay dos pixeles por unidad independiente del dispositivo, lo que significa que el mapa de bits real que se muestra en pantalla que no es ICON-Pequeño-40.png pero IconSmall-40@2x.png, que es dos veces 40, o 80 píxeles cuadrados.

Si en lugar de ejecutar el programa en el iPhone 6 Plus, que cuenta con una unidad independiente del dispositivo igual a tres píxeles-usted ver de nuevo un tamaño de 40 píxeles prestado, lo que significa que el mapa de bits Icon-Small-40@3x.png se visualiza. Ahora probarlo en el simulador de iPad 2. El iPad 2 tiene un tamaño de pantalla de tan sólo 768 × 1024, y las unidades independientes del dispositivo son los mismos que píxeles. Ahora se muestra el mapa de bits del icono-Pequeño-40.png, y el tamaño rendido todavía es 40 píxeles.

Esto es lo que quieres. ¿Quieres ser capaz de controlar el tamaño de los mapas de bits prestados en unidades independientes del dispositivo porque así es como se puede lograr tamaños de mapa de bits sensiblemente similares en diferentes dispositivos y plataformas. Cuando se especifica el mapa de bits del icono-Pequeño-40.png, desea que ese mapa de bits que se representa como 40 unidades o independientes del dispositivo sobre todos los dispositivos IOS-pulgadas en una cuarta parte. Pero si el programa se está ejecutando en un dispositivo de Apple Retina, que no quiere un mapa de bits de 40 pixeles cuadrados se extendía a ser de 40 unidades independientes del dispositivo. Por fidelidad visual máxima, que desea una mayor resolución de mapa de bits está representada, con un mapeo uno a uno de los pixeles de mapas de bits para detectar píxeles.

Si nos fijamos en el Android recursos directorio, encontrará cuatro versiones diferentes de un mapa de bits llamado icon.png. Estos se almacenan en diferentes subcarpetas de recursos:

- dibujable / icon.png - cuadrado 72 píxeles
- dibujable-IPAP / icon.png - 72 píxeles cuadrados
- dibujable-xdpi / icon.png - 96 píxeles cuadrados
- dibujable-xxdpi / icon.png - 144 píxeles cuadrados

Independientemente del dispositivo Android, el ícono se representa con un tamaño de 48 unidades independientes del dispositivo. En el Nexus 5 utilizado en la captura de pantalla, hay tres pixeles a la unidad independiente del dispositivo, lo que significa que el mapa de bits en realidad aparece en esa pantalla es el uno en el **dibujable-xxdpi** carpeta, que es de 144 píxeles cuadrados.

Lo bueno de tanto iOS y Android es que sólo se necesita suministrar mapas de bits de varios Sizes- y darles los nombres correctos o almacenarlos en las carpetas correctas y el sistema operativo elige una imagen óptima para la resolución particular del dispositivo.

La plataforma Windows Runtime tiene una instalación similar. En el **UWP** proyectar verá los nombres de archivo que incluyen escala-200; por ejemplo, Square150x150Logo.scale-200.png. El número después de la palabra *escala* es un porcentaje, y aunque el nombre del archivo parece indicar que se trata de un mapa de bits de 150 x 150, la imagen es

en realidad dos veces más grande: 300 × 300. En el **ventanas** proyecto verá los nombres de archivo que incluyen escala-100 y en el **WinPhone** proyecto que verá la escala-240.

Sin embargo, se ha visto que Xamarin.Forms en el tiempo de ejecución de Windows muestra los mapas de bits en sus tamaños independientes del dispositivo, y usted todavía tiene que tratar a las plataformas de Windows un poco diferente. Pero en las tres plataformas se puede controlar el tamaño de los mapas de bits en unidades independientes del dispositivo.

Al crear sus propias imágenes específicas de la plataforma, sigue las instrucciones en las tres secciones siguientes.

mapas de bits independientes del dispositivo para iOS

El esquema de nombres IOS para mapas de bits implica un sufijo en el nombre de archivo. El sistema operativo obtiene un mapa de bits particular con el nombre de archivo subyacente basado en la resolución de píxeles aproximada del dispositivo:

- No sufijo para 160 dispositivos DPI (1 pixel a la unidad independiente del dispositivo)
- @ Sufijo 2x para 320 dispositivos DPI (2 píxeles a la DIU)
- @ 3x sufijo: 480 dispositivos DPI (3 píxeles a la DIU)

Por ejemplo, supongamos que desea un myimage.jpg mapa de bits llamado para aparecer como aproximadamente una pulgada cuadrada en la pantalla. Usted debe suministrar tres versiones de este mapa de bits:

- Myimage.jpg - 160 píxeles cuadrados
- MyImage@2x.jpg - 320 píxeles cuadrados
- MyImage@3x.jpg - 480 píxeles cuadrados

El mapa de bits se hacen como 160 unidades independientes del dispositivo. Para tamaños más pequeños prestados de una pulgada, disminuir los píxeles proporcionalmente.

Al crear estos mapas de bits, comience con el más grande. A continuación, puede utilizar cualquier utilidad de mapa de bits de edición para reducir el tamaño de píxel. Para algunas imágenes, es posible que desee ajustar con precisión o completamente redibujar las versiones más pequeñas.

Como habrá notado al examinar los diversos archivos de iconos que incluye la plantilla Xamarin.Forms con el proyecto de iOS, no todos los mapas de bits vienen en las tres resoluciones. Si IOS no puede encontrar un mapa de bits con el sufijo particular se quiere, se caerá de nuevo y utilizar uno de los otros, la ampliación del mapa de bits arriba o hacia abajo en el proceso.

mapas de bits independientes del dispositivo para Android

Para Android, mapas de bits se almacenan en varias subcarpetas de recursos que corresponden a una resolución de píxel de la pantalla. Android define seis nombres de directorio diferentes para seis diferentes niveles de resolución del dispositivo:

- **dibujable-LDPI** (bajo DPI) para 120 dispositivos DPI (0,75 píxeles a la DIU)

- **dibujable-MDPI** (medio) para los dispositivos 160 DPI (1 pixel a la DIU)
- **dibujable-IPAP** (alto) para 240 dispositivos DPI (1,5 píxeles a la DIU))
- **dibujable-xhdpi** (extra-alta) para 320 dispositivos DPI (2 píxeles a la DIU)
- **dibujable-xxhdpi** (extra extra alto) para 480 dispositivos DPI (3 píxeles a la DIU)
- **dibujable-xxxhdpi** (tres altos extra) para los dispositivos 640 ppp (4 píxeles a la DIU)

Si quieres un myimage.jpg mapa de bits llamado para rendir como un cuadrado de una pulgada en la pantalla, puede suministrar hasta seis versiones de este mapa de bits utilizando el mismo nombre en todos estos directorios. El tamaño de este mapa de bits de una pulgada cuadrada en pixeles es igual a la DPI asociado a ese directorio:

- dibujable-LDPI / myimage.jpg - 120 píxeles cuadrados
- dibujable-MDPI / myimage.jpg - 160 píxeles cuadrados
- dibujable-IPAP / myimage.jpg - 240 píxeles cuadrados
- dibujable-xhdpi / myimage.jpg - 320 píxeles cuadrada
- dibujable-xxhdpi / myimage.jpg - 480 píxeles cuadrados
- dibujable-xxxhdpi / myimage.jpg - 640 píxeles cuadrados

El mapa de bits se hacen como 160 unidades independientes del dispositivo.

Usted no está obligado a crear mapas de bits para los seis resoluciones. El proyecto Android creado por la plantilla incluye sólo **Xamarin.Forms dibujable-IPAP, dibujable-xhdpi, y dibujable-xxdipi, así como una innecesaria dibujable carpeta, sin sufijo. Estos abarcan** los dispositivos más comunes. Si el sistema operativo Android no encuentra un mapa de bits de la resolución deseada, se caerá de nuevo a un tamaño que está disponible y reduce la escala.

mapas de bits independientes del dispositivo de tiempo de ejecución para las plataformas Windows

El tiempo de ejecución de Windows es compatible con un esquema de mapa de bits de nombres que permite incorporar un factor de escala de pixeles por unidad independiente del dispositivo expresado como un porcentaje. Por ejemplo, para un mapa de bits de una pulgada cuadrada dirigido a un dispositivo que tiene dos pixeles a la unidad, utilice el nombre:

- MyImage.scale-200.jpg - 320 píxeles cuadrada

La documentación de Windows no es clara acerca de los porcentajes reales que puede utilizar. Cuando la construcción de un programa, a veces verá mensajes de error en el **Salida** ventana con respecto a los porcentajes que no están soportadas en la plataforma en particular.

Sin embargo, dado que Xamarin.Forms muestra mapas de bits de Windows en tiempo de ejecución en sus tamaños independientes del dispositivo, esta instalación es de uso limitado en estos dispositivos.

Veamos un programa que realmente hace suministrar mapas de bits personalizados de varios tamaños para las tres plataformas. Estos mapas de bits están destinados a ser prestados aproximadamente una pulgada cuadrada, que es aproximadamente la mitad de la anchura de la pantalla del teléfono en el modo vertical.

Esta **ImageTap** programa crea un par de objetos de botón-como rudimentarios, que no sangrables Ver el texto, pero un mapa de bits. Los dos botones que **ImageTap** crea que podría sustituir a la tradicional **DE ACUERDO** y **Cancelar** botones, pero tal vez quieran usar caras de pinturas famosas de los botones. Tal vez desea que el **DE ACUERDO** botón para mostrar la cara de Venus de Botticelli y la **Cancelar** botón para visualizar el hombre triste de Edvard Munch *El grito*.

En el código de ejemplo para este capítulo es un directorio llamado **Imágenes** que contiene este tipo de imágenes, llamado *Venus_xxx.jpg* y *Scream_xxx.jpg*, donde el xxx indica el tamaño en píxeles. Cada imagen es en ocho tamaños diferentes: 60, 80, 120, 160, 240, 320, 480, y 640 píxeles cuadrados. Además, algunos de los archivos tienen nombres de *Venus_xxx_id.jpg* y *Scream_xxx_id.jpg*. Estas versiones tienen el tamaño de píxel real que se muestra en la esquina inferior derecha de la imagen, de manera que podemos ver en la pantalla de mapa de bits exactamente lo que el sistema operativo ha seleccionado.

Para evitar confusiones, se añadieron los mapas de bits con los nombres originales a la **ImageTap** carpetas de proyectos en primer lugar, y luego fueron renombrados dentro de Visual Studio.

En el **recursos** carpeta del proyecto IOS, se renombró los siguientes archivos:

- *Venus_160_id.jpg* convirtió *Venus.jpg*
- *Venus_320_id.jpg* porque *Venus@2x.jpg*
- *Venus_480_id.jpg* convirtió *Venus@3x.jpg*

Esto se hizo de manera similar para los mapas de bits *Scream.jpg*.

En las distintas subcarpetas del proyecto Android **recursos** carpeta, se renombró los siguientes archivos:

- *Venus_160_id.jpg* convirtió dibujable-MDPI / *Venus.jpg*
- *Venus_240_id.jpg* convirtió dibujable-IPAP / *Venus.jpg*
- *Venus_320_id.jpg* convirtió dibujable-xhdpi / *Venus.jpg*
- *Venus_480_id.jpg* convirtió drawable_xxhdpi / *Venus.jpg*

Y lo mismo para los mapas de bits *Scream.jpg*.

Para el proyecto de Windows Phone 8.1, los archivos *Venus_160_id.jpg* y *Scream_160_id.jpg* fueron copiados a una **imágenes** carpeta y renombrado *Venus.jpg* y *Scream.jpg*.

El proyecto de Windows 8.1 crea un archivo ejecutable que se ejecuta en los teléfonos no, sino en tablas y ordenadores de sobremesa. Estos dispositivos han asumido tradicionalmente una resolución de 96 unidades a la pulgada, lo que los archivos *Venus_100_id.jpg* y *Scream_100_id.jpg* fueron copiados a una **imágenes** carpeta y renombrado *Venus.jpg* y *Scream.jpg*.

El proyecto UWP se dirige a todos los factores de forma, por lo que varios mapas de bits fueron copiados a una **Imágenes** carpeta y renombrado de manera que los mapas de bits 160 píxeles cuadrados serían utilizadas en los teléfonos, y los mapas de bits cuadrados de 100 píxeles se utilizarían en las tabletas y las pantallas de escritorio:

- Venus_160_id.jpg convirtió Venus.scale-200.jpg
- Venus_100_id.jpg convirtió Venus.scale-100.jpg

Y lo mismo para los mapas de bits Scream.jpg.

Cada uno de los proyectos requiere una diferente **construir Acción** para estos mapas de bits. Esto se debe establecer de forma automática cuando se agrega a los archivos de los proyectos, sino que debe de verificar dos veces para asegurarse de que la **construir Acción** se ha establecido correctamente:

- iOS: **BundleResource**
- Androide: **AndroidResource**
- Ventanas de tiempo de ejecución: **Contenido**

Usted no tiene que memorizar estos. En caso de duda, simplemente marque la **construir Acción** para los mapas de bits incluidos en la plantilla de solución Xamarin.Forms en los proyectos de plataforma.

El archivo XAML para el **ImageTap** programa pone cada uno de los dos **Imagen** elementos en una **ContentView** que es de color blanco a partir de un estilo implícita. Esta blanco **ContentView** está cubierto completamente por el **Imagen**, pero (como se verá) que entra en juego cuando el programa parpadea la imagen para indicar que ha sido intervenido.

```
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
  xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
  x: Class = " ImageTap.ImageTapPage " >

  < StackLayout >
    < StackLayout.Resources >
      < ResourceDictionary >
        < Estilo Tipo de objetivo = " ContentView " >
          < Setter Propiedad = " Color de fondo " Valor = " Blanco " />
          < Setter Propiedad = " HorizontalOptions " Valor = " Centrar " />
          < Setter Propiedad = " VerticalOptions " Valor = " CenterAndExpand " />
        </ Estilo >
      </ ResourceDictionary >
    </ StackLayout.Resources >

    < ContentView >
      < Imagen >
        < Fuente de imagen >
          < OnPlatform x: TypeArguments = " Fuente de imagen "
            iOS = " Venus.jpg "
            Androide = " Venus.jpg "
            WinPhone = " imágenes / Venus.jpg " />
        </ Fuente de imagen >
      </ Imagen >
    </ ContentView >
  </ StackLayout >
</ Pagina de contenido >
```

```

< Image.GestureRecognizers >
    < TapGestureRecognizer aprovechado = " OnImageTapped " />
</ Image.GestureRecognizers >

< Imagen >
</ ContentView >

< ContentView >
    < Imagen >
        < Fuente de imagen >
            < OnPlatform x: TypeArguments = " Fuente de imagen "
                iOS = " Scream.jpg "
                Androide = " Scream.jpg "
                WinPhone = " Imágenes / Scream.jpg " />
        </ Fuente de imagen >

        < Image.GestureRecognizers >
            < TapGestureRecognizer aprovechado = " OnImageTapped " />
        </ Image.GestureRecognizers >
    </ Imagen >
</ ContentView >

< Etiqueta x: Nombre = " etiqueta "
    Tamaño de fuente = " Medio "
    HorizontalOptions = " Centrar "
    VerticalOptions = " CenterAndExpand " />

</ StackLayout >
</ Pagina de contenido >

```

Los usos de archivos XAML OnPlatform para seleccionar los nombres de archivo de los recursos de la plataforma. Observe que el **x: TypeArguments** atributo de la **OnPlatform** se establece en **Fuente de imagen** porque este tipo debe coincidir exactamente con el tipo de la propiedad **de destino**, que es el **Fuente** propiedad de **Imagen**. **Fuente de imagen** define una conversión implícita de cuerda a sí mismo, de modo que especifica los nombres de archivo es suficiente. (La lógica para esta conversión implícita comprueba primero si la cadena tiene un prefijo URI. Si no, se supone que la cadena es el nombre de un archivo incrustado en el proyecto de plataforma).

Si desea evitar el uso **OnPlatform** enteramente en los programas que utilizan los mapas de bits de la plataforma, puede poner los mapas de bits de Windows en el directorio raíz del proyecto en lugar de en una carpeta.

Un toque en cualquiera de estos botones hace dos cosas: La **aprovechado** establece el **manejador Opacidad** propiedad de la **Imagen** a 0,75, lo que resulta en que revela parcialmente el blanco **ContentView** fondo y la simulación de un flash. Un temporizador restaura la **Opacidad** en el valor predeterminado de una décima de segundo más tarde. los **aprovechado** manejador también muestra el tamaño de la mostrada **Imagen** elemento:

```

pública clase parcial ImageTapPage : Pagina de contenido
{
    pública ImageTapPage ()
    {
        InitializeComponent ();
    }

    vacío OnImageTapped ( objeto remitente, EventArgs args )

```

```

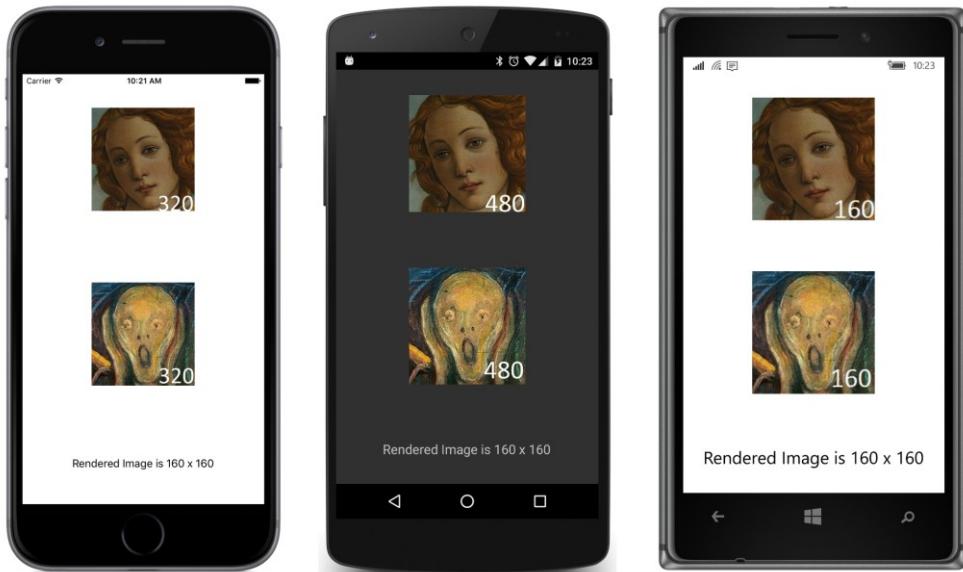
{
    Imagen image = (Imagen)remitente;
    image.Opacity = 0,75;

    Dispositivo.StartTimer ( Espacio de tiempo.FromMilliseconds (100), () =>
    {
        image.Opacity = 1;
        falso retorno ;
    });
}

label.text = Cuerda.Format("Imagen representada es {0} x {1}",
                           image.Width, image.Height);
}
}

```

Que el tamaño dictada en comparación con los tamaños de pixel en los mapas de bits confirma que las tres plataformas de hecho han seleccionado el mapa de bits óptimo:



Estos botones ocupan aproximadamente la mitad del ancho de la pantalla en las tres plataformas. Este dimensionamiento se basa totalmente en el tamaño de los propios mapas de bits, sin ninguna información adicional de calibres en el código o marcado.

Barras de herramientas y sus iconos

Uno de los principales usos de los mapas de bits en la interfaz de usuario es la barra de herramientas Xamarin.Forms, que aparece en la parte superior de la página en iOS y dispositivos Android y en la parte inferior de la página en los dispositivos Windows Phone. Elementos de la barra son hacer tapping y fuego hecho clic eventos como mucho Botón.

No hay ninguna clase para la propia barra. En su lugar, se agrega objetos de tipo `ToolbarItem` al `ToolbarItems` propiedad de colección definida por Página.

los `ToolbarItem` clase no se deriva de `Ver me gusta Etiqueta y Botón`. En su lugar, se deriva de Elemento por medio de `MenuItemBase` y Opción del menú. (Opción del menú se utiliza solamente en conexión con el `TableView` y no será discutido hasta el capítulo 19.) para definir las características de un elemento de barra de herramientas, utilizar las siguientes propiedades:

- **Texto** - el texto que pueda aparecer (dependiendo de la plataforma y Orden)
- **Icono** - un `FileImageSource` objeto referencia a un mapa de bits a partir del proyecto de plataforma
- **Orden** - un miembro de la `ToolbarItemOrder` enumeración: De manera predeterminada, primaria, o Secundario

También hay una `Nombre` propiedad, pero simplemente duplica la `Texto` propiedad y deben considerarse obsoleta.

los `Orden` propiedad determina si el `ToolbarItem` aparece como una imagen (Primario) o texto (Secundario). 10 Las plataformas móviles Windows Phone y Windows están limitados a cuatro Primario artículos, y tanto el iPhone y los dispositivos Android comienza a recibir llenas de más que eso, así que es una limitación razonable. **Adicional Secundario Los artículos son sólo texto. En el iPhone que aparecen debajo de la Primario artículos; en Android y Windows Phone no se ven en la pantalla hasta que el usuario toca una elipse vertical u horizontal.**

los `Icono` propiedad es crucial para Primario artículos, y el `Texto` propiedad es crucial para Secundario artículos, pero el tiempo de ejecución de Windows utiliza también `Texto` para mostrar un atisbo de texto corto por debajo de los iconos para Primario artículos.

Cuando el `ToolbarItem` se toca, se dispara una hecho clic evento. `ToolbarItem` también tiene Mando y `CommandParameter` propiedades como la `Botón`, pero estos son para los propósitos de enlace de datos y se demostrarán en un capítulo posterior.

los `ToolbarItems` colección definida por Página es de tipo `IList <ToolbarItem>`. Una vez que se agrega una `ToolbarItem` a esta colección, la `ToolbarItem` propiedades no se pueden cambiar. Los valores de propiedades en cambio se utilizan internamente para construir objetos específicos de la plataforma.

Puedes añadir `ToolbarItem` objetos a una `Pagina de contenido` en Windows Phone, iOS y Android, pero restringen las barras de herramientas a una `NavigationPage` o a una página navegado a partir de una `NavigationPage`. Afortunadamente, este requisito no significa que todo el tema de la página de navegación necesita ser discutido antes de poder utilizar la barra de herramientas. crear instancias de una `NavigationPage` en lugar de un `Pagina de contenido` implica simplemente llamando al `NavigationPage` constructor con la recién creada `Pagina de contenido` objeto en el Aplicación clase.

los `ToolbarDemo` programa reproduce la barra de herramientas que viste en las capturas de pantalla en el capítulo 1. La `ToolbarDemoPage` deriva de `Pagina de contenido`, pero el Aplicación clase pasa el `ToolbarDemoPage` oponerse a una `NavigationPage` constructor:

```
clase pública Aplicación : Solicitud
{
    público App () {
        MainPage = nuevo NavigationPage ( nuevo ToolbarDemoPage () );
    }
    ...
}
```

Eso es todo lo que es necesario para obtener la barra de herramientas para trabajar en iOS y Android, y tiene algunas otras consecuencias también. Un título que se puede establecer con el Título propiedad de Página se muestra en la parte superior de las pantallas de iOS y Android, y el ícono de la aplicación también se visualiza en la pantalla de Android. Otro resultado del uso NavigationPage es que ya no es necesario para establecer un poco de relleno en la parte superior de la pantalla de iOS. La barra de estado está ahora fuera de la gama de la página de la aplicación.

Quizás el aspecto más difícil de usar ToolbarItem es el montaje de las imágenes de mapa de bits para el ícono propiedad. Cada plataforma tiene diferentes requisitos para la composición de color y el tamaño de estos iconos, y cada plataforma tiene algo diferentes convenciones para las imágenes. El ícono estándar para Compartir, por ejemplo, es diferente en las tres plataformas.

Por estas razones, tiene sentido para cada uno de los proyectos de plataforma para tener su propia colección de íconos de la barra, y por eso ícono es de tipo FileImageSource.

Vamos a comenzar con las dos plataformas que proporcionan colecciones de íconos adecuados para ToolbarItem.

Iconos para Android

El sitio web de Android tiene una colección descargable de íconos de la barra en la siguiente dirección:

<http://developer.android.com/design/downloads>

Descargar el archivo ZIP identificado como Barra de acción Icon Pack.

Los contenidos descomprimidos se organizan en dos directorios principales: (**Core_Icons** 23 imágenes) y **Iconos de la barra de acción** (144 imágenes). Estos son todos los archivos PNG, y el Acción iconos de la barra vienen en cuatro tamaños diferentes, indicados por el nombre del directorio:

- **dibujable-MDPI** (medio DPI) - 32 píxeles cuadrados
- **dibujable-IPAP** (alta DPI) - 48 píxeles cuadrados
- **dibujable-xhdpi** (alta PPP adicional) - 64 píxeles cuadrados
- **dibujable-xxhdpi** (muy alta DPI adicional) - 96 píxeles cuadrados

Estos nombres de directorio son los mismos que la recursos carpetas en su proyecto Android e implican que los íconos de la barra de herramientas hacen que a las 32 unidades independientes del dispositivo, o alrededor de una quinta parte de una pulgada.

los **Core_Icons** carpeta también organiza sus íconos en cuatro directorios con las mismas cuatro tamaños, pero estos directorios se denominan **MDPI**, **IPAP**, **xdpi**, y **sin escala**.

los **Acción Iconos de la barra** carpeta tiene una organización directorio adicional usando los nombres **holo_dark** y **holo_light**:

- **holo_dark** -white imagen en primer plano sobre un fondo transparente
- **holo_light** -black imagen en primer plano sobre un fondo transparente

La palabra "holo" es sinónimo de "holográfico" y se refiere al nombre de Android utiliza para sus temas de color. Aunque el **holo_light** iconos son mucho más fáciles de ver en **Descubridor y Explorador de Windows**, para la mayoría de los propósitos (y especialmente para los elementos de la barra) se debe utilizar el **holo_dark** iconos. (Por supuesto, si usted sabe cómo cambiar el tema de la aplicación en el archivo `AndroidManifest.xml`, entonces es probable que también se sabe que usar la otra colección de iconos).

los **Core_Icons** carpeta contiene sólo los iconos con los primeros planos blancos sobre un fondo transparente.

Para el **ToolbarDemo** programa, tres iconos fueron seleccionados de la **holo_dark** directorio en las cuatro resoluciones. Estos se copian en las subcarpetas correspondientes de la **recursos** directorio en el proyecto Android:

- Desde el **01_core_edit** directorio, los archivos denominados `ic_action_edit.png`
- Desde el **01_core_search** directorio, los archivos denominados `ic_action_search.png`
- Desde el **01_core_refresh** directorio, los archivos denominados `ic_action_refresh.png`

Compruebe las propiedades de estos archivos PNG. Deben tener una construir **Acción de AndroidResource**.

Iconos para las plataformas de Windows en tiempo de ejecución

Si usted tiene una versión de instalación para Windows Phone 8 Visual Studio, se puede encontrar una colección de archivos PNG adecuado para **ToolbarItem** en el siguiente directorio en el disco duro:

C:\ Archivos de programa (x86) \ Microsoft SDKs \ Windows Phone \ v8.0 \ Icons

Puede utilizar estas para todas las plataformas de Windows en tiempo de ejecución.

Hay dos subdirectorios, **Oscuro** y **Ligero**, conteniendo cada uno los mismos 37 imágenes. Al igual que con Android, los iconos de la **Oscuro** directorio de tener los primeros planos blancos sobre fondo transparente, y los iconos de la **Ligero** directorio tiene los primeros planos negros sobre fondos transparentes. Debe utilizar los de la **Oscuro** directorio para Windows Phone 8.1 y el **Ligero** directorio para Windows 10 móvil.

Las imágenes son un cuadrado de 76 píxeles, pero uniformes han sido diseñados para aparecer dentro de un círculo. De hecho, uno de los archivos se llama `basecircle.png`, que puede servir como una guía si desea diseñar su propia, por lo que en realidad sólo hay 36 iconos utilizables en la colección y un par de ellos son los mismos.

En general, en un proyecto de Windows en tiempo de ejecución, los archivos como estos se almacenan en el **Bienes** carpeta (que ya existe en el proyecto) o una carpeta con el nombre **Imágenes**. Los siguientes mapas de bits se añadieron a una **imágenes** carpeta en las tres plataformas de Windows:

- edit.png
- feature.search.png
- refresh.png

Para la plataforma Windows 8.1 (pero no el teléfono de Windows 8.1 de la plataforma), se necesitan iconos para todos los elementos de la barra, por lo que se añadieron los siguientes mapas de bits a la **ícones** carpeta de ese proyecto:

- Icon1F435.png
- Icon1F440.png
- Icon1F52D.png

Estos se generaron en un programa de Windows desde la fuente Segoe UI símbolo, que es compatible con caracteres emoji. El número hexadecimal de cinco dígitos en el nombre del archivo es el ID de Unicode para los caracteres.

Cuando se agrega iconos para un proyecto de Windows en tiempo de ejecución, asegúrese de que la **construir Acción es Contenido**.

Los iconos de los dispositivos IOS

Esta es la plataforma más problemático para ToolbarItem. Si está programando directamente para la API de iOS nativa, un grupo de constantes que se pueda seleccionar una imagen para UIBarButtonItem, que es la aplicación de iOS subyacente de ToolbarItem. Sin embargo, para los Xamarin.Forms ToolbarItem, que necesita para obtener los iconos de otra fuente, tal vez una colección de licencias como la de glyphish.com-o hacer su propio.

Para obtener los mejores resultados, debe suministrar dos o tres archivos de imagen para cada elemento de la barra de herramientas en el **recursos** carpeta. Una imagen con un nombre de archivo tal como image.png debe ser de 20 píxeles cuadrados, mientras que la misma imagen también debe ser suministrado en una dimensión de 40 píxeles cuadrados con el nombre image@2x.png y como un mapa de bits 60pixel cuadrados llamado imagen @ 3x.png.

Aquí hay una colección de iconos libres, sin restricciones de uso se utiliza para el programa en el Capítulo 1 y para el **ToolbarDemo** programa en este capítulo:

<http://www.smashingmagazine.com/2010/07/14/gcons-free-all-purpose-icons-for-designers-and-developers-100-icons-psd/>

Sin embargo, son uniformemente 32 píxeles cuadrados, y algunos básicos faltan. En cualquier caso, los tres mapas de bits siguientes se copian en el **recursos** carpeta en el proyecto IOS bajo el supuesto de que se escalarán adecuadamente:

- edit.png
- Search.png
- reload.png

Otra opción es utilizar los iconos de Android a partir de la **holo_light** directorio y escalar la imagen más grande para los diferentes tamaños de iOS.

Para los iconos de barra de herramientas en un proyecto de iOS, el **construir Acción** debe ser **BundleResource**.

Aquí está la **ToolbarDemo** archivo XAML que muestra las diversas ToolbarItem objetos añadidos a la ToolbarItems colección de la página. los x: TypeArguments para atribuir OnPlatform debe ser FileImageSource en este caso, ya que ese es el tipo de la Icono propiedad de ToolbarItem. Los tres elementos marcados como Secundario sólo tienen la Texto conjunto de propiedades y no el Icono propiedad.

El elemento raíz tiene una Título conjunto de propiedades en la página. Esto se muestra en las pantallas de iOS y Android cuando la página se instancia como una NavigationPage (o navegado a partir de una Navegación-Página):

```
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns:x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " ToolbarDemo.ToolbarDemoPage "
    Título = " Barra de herramientas de demostración ">

    < Etiqueta x: Nombre = " etiqueta "
        Tamaño de fuente = " Medio "
        HorizontalOptions = " Centrar "
        VerticalOptions = " Centrar " />

    < ContentPage.ToolbarItems >
        < ToolbarItem Texto = " editar "
            Orden = " Primario "
            hecho clic = " OnToolbarItemClicked " >
            < ToolbarItem.Icon >
                < OnPlatform x: TypeArguments = " FileImageSource "
                    iOS = " edit.png "
                    Androide = " ic_action_edit.png "
                    WinPhone = " Imágenes / edit.png " />
            </ ToolbarItem.Icon >
        </ ToolbarItem >
        < ToolbarItem Texto = " buscar "
            Orden = " Primario "
            hecho clic = " OnToolbarItemClicked " >
            < ToolbarItem.Icon >
                < OnPlatform x: TypeArguments = " FileImageSource "
                    iOS = " Search.png "
                    Androide = " ic_action_search.png "
                    WinPhone = " Imágenes / feature.search.png " />
            </ ToolbarItem.Icon >
        </ ToolbarItem >
        < ToolbarItem Texto = " refrescar "
            Orden = " Primario "
            hecho clic = " OnToolbarItemClicked " >
            < ToolbarItem.Icon >
                < OnPlatform x: TypeArguments = " FileImageSource "
```

```

        iOS = "reload.png"
        Androide = "ic_action_refresh.png"
        WinPhone = "Imágenes / refresh.png" />
    </ToolbarItem.Icon>
</ToolbarItem>

<ToolbarItem Texto = "explorar"
    Orden = "Secundario"
    hecho clic = "OnToolbarItemClicked">
    <ToolbarItem.Icon>
        <OnPlatform x: TypeArguments = "FileImageSource"
            WinPhone = "Imágenes / icon1F52D.png" />
    </ToolbarItem.Icon>
</ToolbarItem>

<ToolbarItem Texto = "descubrir"
    Orden = "Secundario"
    hecho clic = "OnToolbarItemClicked">
    <ToolbarItem.Icon>
        <OnPlatform x: TypeArguments = "FileImageSource"
            WinPhone = "Imágenes / icon1F440.png" />
    </ToolbarItem.Icon>
</ToolbarItem>

<ToolbarItem Texto = "evolucionar"
    Orden = "Secundario"
    hecho clic = "OnToolbarItemClicked">
    <ToolbarItem.Icon>
        <OnPlatform x: TypeArguments = "FileImageSource"
            WinPhone = "Imágenes / icon1F435.png" />
    </ToolbarItem.Icon>
</ToolbarItem>
</ContentPage.ToolbarItems>
</Pagina de contenido>

```

Aunque el OnPlatform elemento implica que existen los iconos secundarias para todas las plataformas de Windows en tiempo de ejecución, no lo hacen, pero no pasa nada malo si el archivo de ícono en particular no se encuentra en el proyecto.

Todos hecho clic eventos tienen el mismo controlador asignado. Puede utilizar los controladores únicos para los artículos, por supuesto. Este controlador sólo muestra el texto de la ToolbarItem usando el centrado Etiqueta:

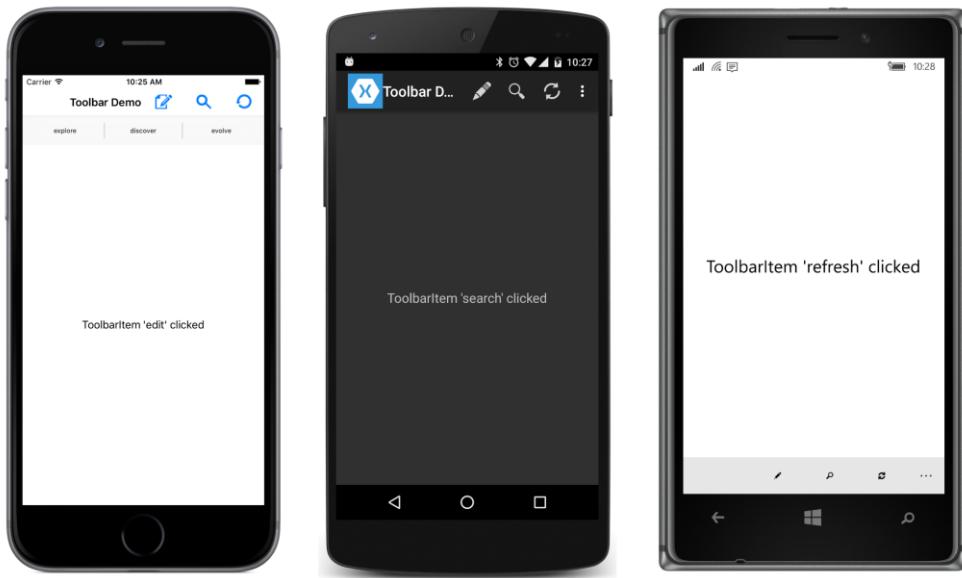
```

pública clase parcial ToolbarDemoPage : Pagina de contenido
{
    pública ToolbarDemoPage ()
    {
        InitializeComponent ();
    }

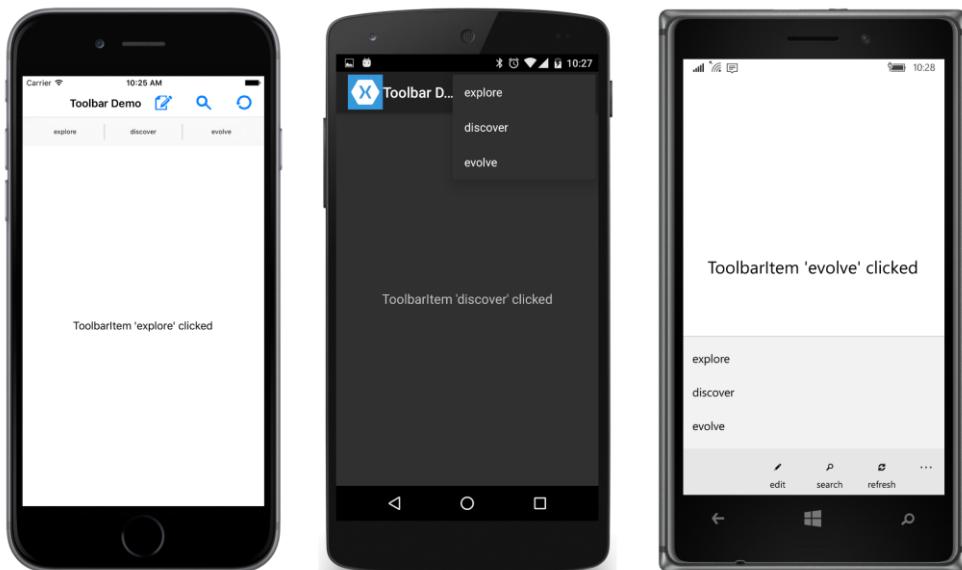
    vacío OnToolbarItemClicked ( objeto remitente, EventArgs args)
    {
        ToolbarItem ToolbarItem = ( ToolbarItem )remitente;
        label.text = "ToolbarItem" + + ToolbarItem.Text " Clic" ;
    }
}

```

Las capturas de pantalla muestran los elementos de la barra icono (y para iOS, los elementos de texto) y el centraron Etiqueta con el elemento más recientemente hecho clic:



Si toca los puntos suspensivos en la parte superior de la pantalla Android o los puntos suspensivos en la esquina inferior derecha de la pantalla móvil de Windows 10, los elementos de texto se muestran y, además, los elementos de texto asociados a los iconos se muestran también en Windows 10 móvil:



Independientemente de la plataforma, la barra de herramientas es la forma estándar de añadir comandos comunes a una aplicación de teléfono.

imágenes de los botones

Botón define una Imagen propiedad de tipo FileImageSource que se puede utilizar para suministrar una pequeña imagen suplementaria que se muestra a la izquierda del texto del botón. Esta característica es *no* destinado a un botón de sólo imagen; si eso es lo que desea, el ImageTap programa en este capítulo es un buen punto de partida.

Desea que las imágenes sean de alrededor de una quinta parte de pulgada de tamaño. Eso significa que usted quiere que ellos hacen a las 32 unidades independientes del dispositivo y aparecen en el contexto de la Botón. Para iOS y el UWP, eso significa que una imagen en negro sobre un fondo blanco o transparente. Para Android, Windows 8.1 y Windows Phone 8.1, tendrá una imagen en blanco contra un fondo transparente.

Todos los mapas de bits en el ButtonImage proyecto son de la Barra de acciones directorio de la Android iconos de diseño la recogida y el 03_rating_good y 03_rating_bad subdirectorios. Estos son los "pulgares arriba" e imágenes "No me gusta".

Las imágenes son del IOS holo_light directorio (imágenes negras sobre fondo transparente) con las siguientes conversiones de nombre de archivo:

- dibujable-MDPI / ic_action_good.png No renombrado
- dibujable-xhdpi / ic_action_good.png renombrado a ic_action_good@2x.png

Y lo mismo para ic_action_bad.png.

Las imágenes son de la Android holo_dark directorio (imágenes en blanco sobre fondo transparente) e incluyen los cuatro tamaños de los subdirectorios dibujable-MDPI (32 píxeles cuadrados), dibujable-IPAP (48 píxeles), dibujable-xhdpi (64 pixeles), y dibujable-xxhdpi (96 píxeles cuadrados).

Las imágenes de los diversos proyectos de Windows en tiempo de ejecución son todos de manera uniforme los mapas de bits de 32 píxeles de la dibujable-mdpi directorios.

Aquí está el archivo XAML que establece el Icono propiedad para dos Botón elementos:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
      xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
      x: Class = " ButtonImage.ButtonImagePage " >

    < StackLayout VerticalOptions = " Centrar "
                  Espaciado = " 50 " >

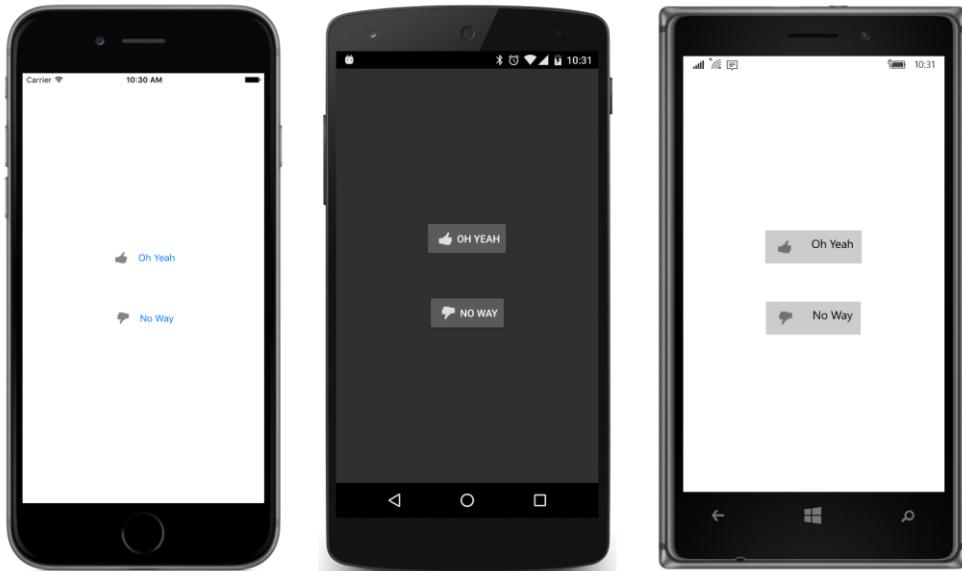
        < StackLayout.Resources >
            < ResourceDictionary >
                < Estilo Tipo de objetivo = " Botón " >
                    < Setter Propiedad = " HorizontalOptions " Valor = " Centrar " />
                    </ Setter>
                </ Estilo >
            </ ResourceDictionary >
        </ StackLayout.Resources >
    </ StackLayout >
</ Página de contenido >
```

```
</Setter>
</Style>
<ResourceDictionary>
</StackLayout.Resources>

<Botón Texto = "Oh si">
    <Button.Image>
        <OnPlatform x: TypeArguments = "FileImageSource "
            iOS = "ic_action_good.png"
            Androide = "ic_action_good.png"
            WinPhone = "Imágenes / ic_action_good.png" />
    </Button.Image>
</Botón>

<Botón Texto = "De ninguna manera ">
    <Button.Image>
        <OnPlatform x: TypeArguments = "FileImageSource "
            iOS = "ic_action_bad.png"
            Androide = "ic_action_bad.png"
            WinPhone = "Imágenes / ic_action_bad.png" />
    </Button.Image>
</Botón>
</StackLayout>
</Página de contenido>
```

Y aquí están:



No es mucho, pero el mapa de bits añade un poco de estilo a la que normalmente sólo texto Botón.

Otro uso importante para los pequeños mapas de bits es el menú de contexto disponible para los elementos de la TableView.

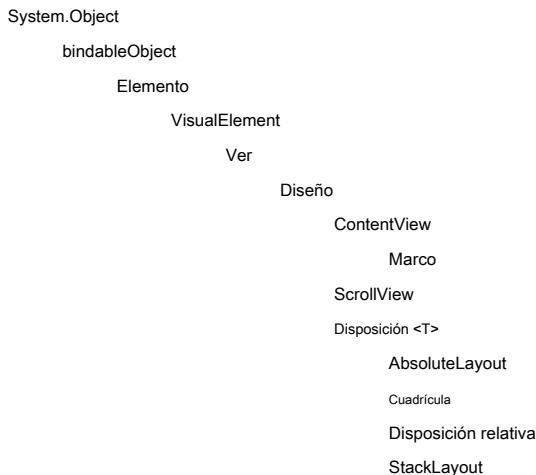
Sin embargo, un requisito previo para que es una exploración profunda de los diversos puntos de vista que contribuyen a la interfaz interactiva de Xamarin.Forms. Eso viene en el capítulo 15.

Pero primero vamos a ver una alternativa a StackLayout que le permite vistas secundarias de posición de una manera completamente flexible.

capítulo 14

disposición absoluta

En Xamarin.Forms, el concepto de diseño abarca todas las formas en que diferentes puntos de vista se pueden montar en la pantalla. Aquí está la jerarquía de clases que muestra todas las clases que se derivan de Diseño:



Ya has visto ContentView, Marco, y ScrollView (todos los cuales tienen una Contenido propiedad que se puede fijar a un niño), y que ha visto StackLayout, que hereda una Niños propiedad de Disposición <T> y muestra sus niños en una pila vertical u horizontal. los Cuadrícula y Relativo-

Diseño la implementación de modelos de diseño un tanto complejos y se exploran en los próximos capítulos. Absoluto-Diseño es el tema de este capítulo.

Al principio, la AbsoluteLayout Parece clase para implementar un diseño bastante primitivo modelo de uno que se remonta a los no tan buenos tiempos de interfaces gráficas de usuario cuando se requiere que los programadores tamaño y la posición individual de cada elemento de la pantalla. Sin embargo, descubrirá que AbsoluteLayout fuera También incorpora un posicionamiento proporcional y la característica que ayuda a dimensionar trae este modelo de diseño antiguo en la era moderna.

Con AbsoluteLayout, muchas de las reglas sobre el diseño que ha aprendido hasta ahora ya no se aplican: la HorizontalOptions y VerticalOptions propiedades que son tan importantes cuando una Ver es el hijo de una Pagina de contenido o StackLayout no tienen absolutamente ningún efecto cuando una Ver es un niño de una AbsoluteLayout. Un programa debe asignar a cada lugar de un niño AbsoluteLayout una ubicación específica en coordenadas independientes del dispositivo. El niño también se le puede asignar un tamaño específico o se permite que el tamaño si.

Puedes usar AbsoluteLayout ya sea en clave o en XAML. De cualquier manera, se encontrará con una característica que

no han visto hasta ahora de que es otra parte del apoyo proporcionado por bindableObject y Bindable-Propiedad. Esta nueva característica es la *adjunta propiedad enlazable*. Este es un tipo especial de propiedad enlazable que se define por una clase (en este caso el AbsoluteLayout) sino que se fija en otros objetos (los hijos de la AbsoluteLayout).

AbsoluteLayout en el código

Se puede añadir una vista de los niños a la Niños cobro de una AbsoluteLayout del mismo modo que con StackLayout:

```
absoluteLayout.Children.Add (niño);
```

Sin embargo, también tiene otras opciones. Los AbsoluteLayout redefine su clase Niños propiedad a ser de tipo AbsoluteLayout.IAbsoluteList <Ver>, que incluye dos adicionales Añadir métodos que permiten especificar la posición del niño en relación con la esquina superior izquierda de la Absoluto-

Diseño. Opcionalmente, puede especificar el tamaño del niño.

Para especificar la posición y tamaño, se utiliza una Rectángulo valor. Rectángulo es una estructura, y se puede crear una Rectángulo valor con un constructor que acepta Punto y tamaño valores:

```
Punto punto = nuevo Punto (X, y);
tamaño size = nuevo tamaño (Anchura, altura);
Rectángulo rect = nuevo Rectángulo (Punto, tamaño);
```

O puede pasar el X, y, ancho, y altura argumentos directamente a una Rectángulo constructor:

```
Rectángulo rect = nuevo Rectángulo (X, y, anchura, altura);
```

A continuación, puede utilizar una alternativa Añadir Método para añadir vistas a la Niños colección de la absoluteLayout:

```
absoluteLayout.Children.Add (niño, rect);
```

Los X y y Los valores indican la posición de la esquina superior izquierda de la vista del niño relativo a la esquina superior izquierda de la AbsoluteLayout los padres en las coordenadas independientes del dispositivo. Si prefiere el niño a sí mismo tamaño, puede utilizar sólo una Punto sin valor tamaño valor:

```
absoluteLayout.Children.Add (niño, punto);
```

He aquí una pequeña demostración en un programa llamado **AbsoluteDemo**:

```
clase pública AbsoluteDemoPage : Página de contenido
{
    público AbsoluteDemoPage ()
    {
        AbsoluteLayout AbsoluteLayout = nuevo AbsoluteLayout
        {
            Relleno = nuevo Espesor (50)
        };
    }
}
```

```
absoluteLayout.Children.Add (
    nuevo BoxView
    {
        color = Color_Acento
    },
    nuevo Rectángulo (0, 10, 200, 5));

absoluteLayout.Children.Add (
    nuevo BoxView
    {
        color = Color_Acento
    },
    nuevo Rectángulo (0, 20, 200, 5));

absoluteLayout.Children.Add (
    nuevo BoxView
    {
        color = Color_Acento
    },
    nuevo Rectángulo (10, 0, 5, 65));

absoluteLayout.Children.Add (
    nuevo BoxView
    {
        color = Color_Acento
    },
    nuevo Rectángulo (20, 0, 5, 65));

absoluteLayout.Children.Add (
    nuevo Etiqueta
    {
        text = "Encabezado con estilo",
        FontSize = 24
    },
    nuevo Punto (30, 25));

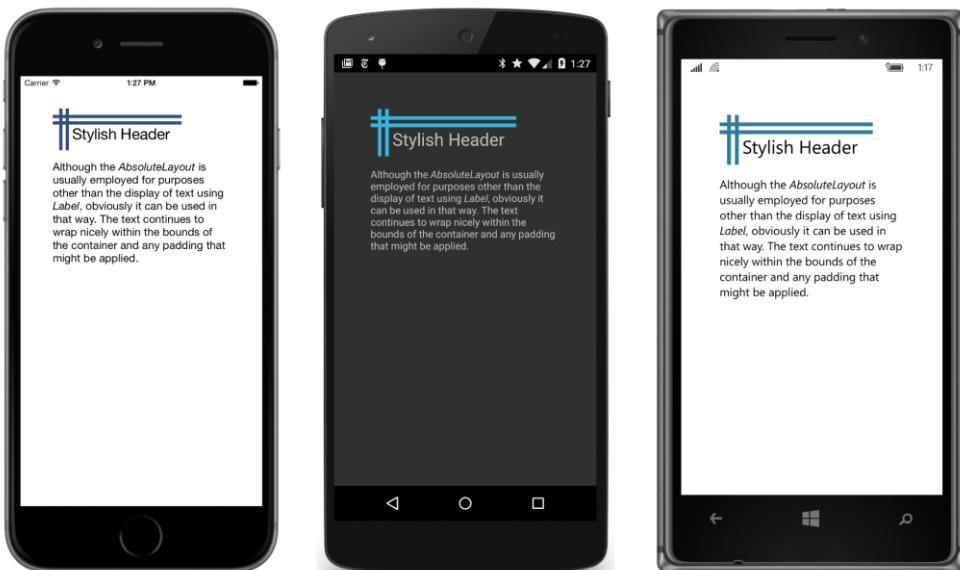
absoluteLayout.Children.Add (
    nuevo Etiqueta
    {
        FormattedText = nuevo FormattedString
        {
            Los tramos =
            {
                nuevo Lapso
                {
                    text = "Aunque el "
                },
                nuevo Lapso
                {
                    text = "AbsoluteLayout",
                    FontAttributes = FontAttributes_Itálico
                },
                nuevo Lapso
```

```

    {
        text = "Se emplea normalmente para otros fines" +
            "Que la visualización de texto usando"
    },
    nuevo Lapso
    {
        text = "Etiqueta",
        FontAttributes = FontAttributes .Itálico
    },
    nuevo Lapso
    {
        text = "Es obvio que se puede utilizar de esa manera." +
            "El texto continúa para envolver muy bien" +
            "Dentro de los límites del contenedor" +
            "Y cualquier relleno que podrían ser aplicadas."
    }
}
},
nuevo Punto (0, 80);

esta .Este contenido ha sido = AbsoluteLayout;
}
}
}

Las cuatro BoxView elementos forman un solapamiento patrón cruzado en la parte superior para hacer estellar una cabecera, y luego un párrafo de texto sigue. Las posiciones de programa y los tamaños de todo el BoxView elementos, mientras que se limita a las posiciones de los dos Etiqueta puntos de vista, ya que tamaño de los mismos:
```



Se necesitaba un poco de ensayo y error para obtener los tamaños de los cuatro BoxView elementos y la cabecera

texto sea aproximadamente el mismo tamaño. Pero nótese que la BoxView elementos se superponen: AbsoluteLayout le permite superponer puntos de vista de una manera muy libre que es simplemente imposible con StackLayout (o sin uso de transformaciones, que están cubiertos en un capítulo posterior).

El gran inconveniente de AbsoluteLayout es que se necesita para llegar a la posición coordinada sí mismo o su cálculo en tiempo de ejecución. Todo lo que no dimensionada tal manera explícita como los dos Etiquetas calculará un tamaño por sí mismo cuando la página se presenta. Pero que el tamaño no está disponible hasta entonces. Si quiere añadir otro párrafo después de la segunda Etiqueta, lo coordenadas usaría?

En realidad, puede colocar varios párrafos de texto, poniendo una StackLayout (o una StackLayout dentro de una ScrollView) en el AbsoluteLayout y luego poner el Etiqueta vistas en eso. Los diseños se pueden anidar.

Como se puede suponer, usando AbsoluteLayout es más difícil que usar StackLayout. En general, es mucho más fácil dejar que Xamarin.Forms y el otro Diseño clases manejan gran parte de la complejidad del diseño para usted. Sin embargo, para algunos usos especiales, AbsoluteLayout es ideal.

Al igual que todos los elementos visuales, AbsoluteLayout tiene su HorizontalOptions y VerticalOptions propiedades ajustado a Llenar por defecto, lo que significa que AbsoluteLayout llena su contenedor. Con otros ajustes de HorizontalOptions y VerticalOptions, un AbsoluteLayout tamaños sí con el tamaño de su contenido, pero hay algunas excepciones: Trate de darle la AbsoluteLayout en el AbsoluteDemo

un programa de Color de fondo para que pueda ver exactamente el espacio que ocupa en la pantalla. Normalmente se llena toda la página, pero si se establece la HorizontalOptions y VerticalOptions propiedades de la AbsoluteLayout a Centrar, verá que el tamaño que la AbsoluteLayout calcula por sí mismo incluye el contenido y el relleno, pero sólo una línea del párrafo de texto.

Averiguar tamaños para elementos visuales en una AbsoluteLayout puede ser complicado. Un enfoque simple se demuestra por el ChessboardFixed programa a continuación. El nombre del programa tiene el sufijo Fijo debido a que la posición y el tamaño de todos los cuadrados dentro del tablero de ajedrez se establecen en el constructor. El constructor no puede anticipar el tamaño de la pantalla, por lo que arbitrariamente establece el tamaño de cada cuadrado de 35 unidades, como se indica por la squareSize constante en la parte superior de la clase. Este valor debe ser suficientemente pequeña para el tablero de ajedrez para que quepa en la pantalla de cualquier dispositivo soportado por Xamarin.Forms.

Observe que el AbsoluteLayout se centra por lo que tendrá un tamaño que se adapte a todos sus hijos. La propia placa se le da un color de piel de ante, que es un pálido amarillo-marrón, y luego 32 de color verde oscuro BoxView elementos se muestran en cada otra posición cuadrado:

```
público clase ChessboardFixedPage : Pagina de contenido
{
    público ChessboardFixedPage ()
    {
        const doble squareSize = 35;

        AbsoluteLayout AbsoluteLayout = nuevo AbsoluteLayout
        {
            BackgroundColor = Color .FromRgb (240, 220, 130),
            HorizontalOptions = LayoutOptions .Centrar,
```

```
VerticalOptions = LayoutOptions.Centrar
};

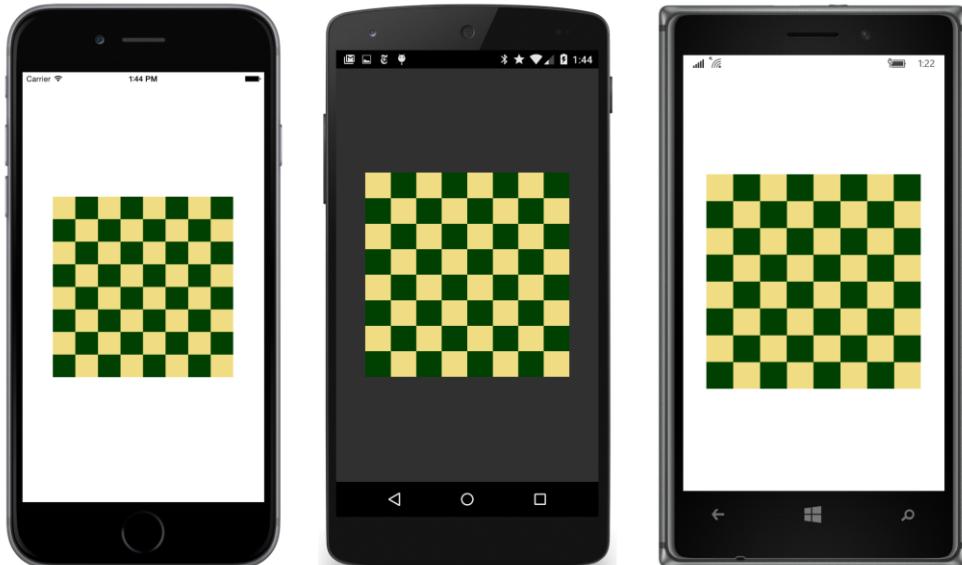
para ( Ent fila = 0; fila <8; fila++)
{
    para ( Ent col = 0; Col <8; col++)
    {
        // Saltar cualquier otro cuadrado.
        Si (((Fila ^ col) y 1) == 0)
            continuar;

        BoxView boxView = nuevo BoxView
        {
            color = Color.FromRgb (0, 64, 0)
        };

        Rectángulo rect = nuevo Rectángulo (Col * squareSize,
                                         * squareSize fila,
                                         squareSize, squareSize);

        absoluteLayout.Children.Add (boxView, rect);
    }
}
esta .Este contenido ha sido = AbsoluteLayout;
}
}
```

El exclusivo o el cálculo de la fila y columna las variables causa una BoxView crearse sólo cuando sea el fila o columna variable es extraño, pero ambos no son impares. Aquí está el resultado:



propiedades enlazables adjuntos

Si quisieramos que este tablero de ajedrez a ser tan grande como sea posible dentro de los límites de la pantalla, tendríamos que añadir el BoxView elementos a la AbsoluteLayout durante el SizeChanged manejador de la página, o la SizeChanged manipulador tendría que encontrar alguna manera de cambiar la posición y el tamaño de la Caja-

Ver elementos ya en el Niños colección.

Ambas opciones son posibles, pero el segundo es preferido porque podemos llenar el Niños colección de la AbsoluteLayout sólo una vez en el constructor del programa y luego ajustar el tamaño y la posición posterior.

En el primer encuentro, la sintaxis que le permite establecer la posición y el tamaño de un niño que ya están en una ABSoluteLayout podría parecer un tanto extraña. Si ver es un objeto de tipo Ver y rect es un Rectan-GLE valor, aquí está la declaración que le dé ver un lugar y tamaño de rect:

```
AbsoluteLayout .SetLayoutParams (vista, rect);
```

Eso no es una instancia de AbsoluteLayout en la que usted está haciendo una SetLayoutParams llamada. No. Eso es un método estático de la AbsoluteLayout clase. Puedes llamar AbsoluteLayout.LayoutParams ya sea antes o después de agregar el ver niño al AbsoluteLayout colección de los niños. De hecho, porque es un método estático, se puede llamar al método antes de la AbsoluteLayout incluso ha creado una instancia! Un caso particular de AbsoluteLayout no está involucrado en absoluto en este SetLayout-
Límites método.

Veamos algo de código que hace uso de este misterioso AbsoluteLayout.LayoutParams

método y luego examinar cómo funciona.

los ChessboardDynamic página del programa constructor utiliza la sencilla Añadir método sin posicionamiento o de encolado para agregar 32 BoxView elementos a la AbsoluteLayout en uno para lazo. Para proporcionar un poco de margen alrededor del tablero de ajedrez, el AbsoluteLayout es un hijo de una ContentView y el relleno se establece en la página. Esta ContentView tiene un SizeChanged manejador a la posición y el tamaño de la AbsoluteLayout-
fuera los niños en función del tamaño del contenedor:

```
público clase ChessboardDynamicPage : Página de contenido
{
    AbsoluteLayout AbsoluteLayout;

    público ChessboardDynamicPage ()
    {
        AbsoluteLayout = nuevo AbsoluteLayout
        {
            BackgroundColor = Color .FromRgb (240, 220, 130),
            HorizontalOptions = LayoutOptions .Centrar,
            VerticalOptions = LayoutOptions .Centrar
        };

        para ( En t i = 0; i <32; i ++)
        {

```

```

BoxView boxView = nuevo BoxView
{
    color = Color.FromRgb(0, 64, 0)
};

absoluteLayout.Children.Add(boxView);
}

ContentView contentView = nuevo ContentView
{
    Content = AbsoluteLayout
};

contentView.SizeChanged += OnContentViewSizeChanged;

esta.Padding = nuevo Espesor(5, Dispositivo.OnPlatform(25, 5, 5), 5, 5);
esta.Este contenido ha sido = contentView;
}

vacío OnContentViewSizeChanged (objeto remitente, EventArgs args)
{
    ContentView contentView = (ContentView)remitente;
    doble squareSize = Mates.min(contentView.Width, contentView.Height) / 8;
    En t index = 0;

    para (En t fila = 0; fila < 8; fila++)
    {
        para (En t col = 0; Col < 8; col++)
        {
            // Saltar cualquier otro cuadrado.
            Si (((Fila ^ col) y 1) == 0)
                continuar;

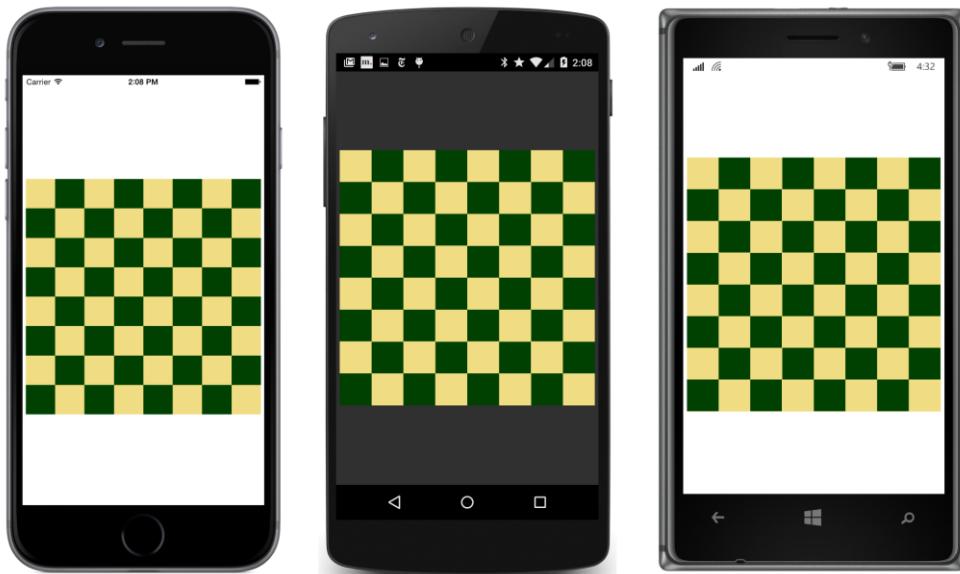
            Ver view = absoluteLayout.Children [índice];
            Rectángulo rect = nuevo Rectángulo (Col * squareSize,
                * squareSize fila,
                squareSize, squareSize);

            AbsoluteLayout.SetLayoutBounds (vista, rect);
            índice++;
        }
    }
}
}

```

los SizeChanged manejador contiene mucho la misma lógica que el constructor en **ChessboardFixed** excepto que el BoxView elementos ya están en el Niños colección de la AbsoluteLayout. Todo lo que es necesario es en la posición y el tamaño de cada uno BoxView cuando el tamaño del ejemplo contenedores cambios-para, durante los cambios de orientación de teléfono. los para bucle concluye con una llamada a la estática Absoluto-Layout.SetLayoutBounds método para cada BoxView con un calculado Rectángulo valor.

Ahora el tablero de ajedrez está dimensionada para ajustarse a la pantalla con un poco de margen:



Obviamente, el misterioso `AbsoluteLayout.SetLayoutBounds` método funciona, pero ¿cómo? ¿Qué hace? ¿Y cómo se las arreglan para hacer lo que hace sin hacer referencia a un particular, Absoluto-Diseño ¿objeto?

los `AbsoluteLayout.SetLayoutBounds` llama que has visto se parece a la siguiente:

```
AbsoluteLayout .SetLayoutBounds (vista, rect);
```

Ese llamado método es exactamente equivalente a la siguiente llamada en la vista del niño:

```
view.SetValue ( AbsoluteLayout .LayoutBoundsProperty, rect);
```

Esto es un Valor ajustado pedir a la vista del niño. Estas dos llamadas a métodos son exactamente equivalentes debido a que el segundo es cómo `AbsoluteLayout` define internamente la `SetLayoutBounds` método estático. `absoluta-`
`luteLayout.SetLayoutBounds` es simplemente un método de acceso directo, y la estática similar, `AbsoluteLay-`
`out.GetLayoutBounds` método es un método abreviado para una `GetValue` llamada.

Usted recordará que `Valor ajustado` y `GetValue` se definen por `bindableObject` y utilizado para implementar propiedades enlazables. A juzgar únicamente por el nombre, `AbsoluteLayout.LayoutBoundsProperty` sin duda parece ser una `BindableProperty` objeto, y que es así. Sin embargo, es un tipo muy especial de propiedad enlazable llamado *adjunta propiedad enlazable*.

propiedades enlazables normales se pueden configurar únicamente en instancias de la clase que define la propiedad o en instancias de una clase derivada. propiedades enlazables adjuntos pueden romper esa regla: propiedades enlazables adjuntos están definidos por una clase, en este caso `AbsoluteLayout` -pero establecido en otro objeto, en este caso, un niño de la `AbsoluteLayout`. La propiedad se dice a veces que *adjunto* para el niño, de ahí el nombre.

El niño de la AbsoluteLayout es ignorante del propósito de la propiedad enlazable unida a su pasado Valor ajustado método, y el niño no hace uso de ese valor en su propia lógica interna. los Valor ajustado Método del niño simplemente guarda el Rectángulo valor en un diccionario mantenida por Enlazar-object dentro del niño, en efecto fijar este valor al niño a ser posiblemente utilizado en algún momento por el padre-la AbsoluteLayout objeto.

Cuando el AbsoluteLayout es trazar sus hijos, puede interrogar al valor de esta propiedad de cada niño llamando a la AbsoluteLayout.GetLayoutBoundsProperty método estático en el niño, que a su vez las llamadas GetValue en el niño con el AbsoluteLayout.LayoutBoundsProperty adjunta propiedad enlazable. La llamada a GetValue Obtiene la Rectángulo valor del diccionario almacenado dentro del niño.

Usted podría preguntarse: ¿Por qué es un proceso tan rotunda necesario para establecer el posicionamiento y la información del apresto en un niño de la AbsoluteLayout? ¿No hubiera sido más fácil para Ver para definir sencilla X, Y, Anchura, y Altura propiedades que una aplicación podría set?

Tal vez, pero esas propiedades serían adecuados sólo para AbsoluteLayout. Cuando se utiliza el Cuadrícula, una aplicación necesita especificar Fila y Columna valores en los niños de la Cuadrícula, y cuando se utiliza una clase de diseño de su propia invención, tal vez algunas otras propiedades son necesarias. propiedades enlazables adjuntos pueden manejar todos estos casos y más.

propiedades enlazables adjuntos son un mecanismo de propósito general que permite a las propiedades definidas por una clase a ser almacenados en los casos de otra clase. Puede definir sus propias propiedades enlazables unidos mediante el uso de métodos de creación estáticos de bindableObject llamado CreateAttached y CreateAttachedReadOnly. (Vas a ver un ejemplo en el capítulo 27, "procesadores personalizados.")

propiedades adjuntas se utilizan sobre todo con las clases de diseño. Como se verá, Cuadrícula define adjunto propiedades enlazables para especificar la fila y columna de cada niño, y Disposición relativa también define adjunto propiedades enlazables.

Antes se vio adicional Añadir métodos definidos por la Niños colección de AbsoluteLayout.

Estas son en realidad implementan utilizando estas propiedades enlazables adjuntos. La llamada

```
absoluteLayout.Children.Add (vista, rect);
```

se implementa como esto:

```
AbsoluteLayout.SetLayoutBounds (vista, rect);
absoluteLayout.Children.Add (vista);
```

los Añadir llamar con sólo una Punto argumento se limita a fijar la posición del niño y permite que el tamaño del niño en sí:

```
absoluteLayout.Children.Add (vista, nuevo Punto (X, y));
```

Esto se implementa con el mismo estática AbsoluteLayout.SetLayoutBounds llamadas, pero utilizando una constante especial para la anchura y la altura de la vista:

```
AbsoluteLayout.SetLayoutBounds (vista,
    nuevo Rectángulo (X, y, AbsoluteLayout.Tamaño automático, AbsoluteLayout.Tamaño automático));
```

```
absoluteLayout.Children.Add (vista);
```

Puede utilizar ese `AbsoluteLayout.AutoSize` constante en su propio código.

De tamaño proporcional y posicionamiento

Como se vio, la `ChessboardDynamic` programa reposiciona y cambia el tamaño de la `BoxView` niños con cálculos basados en el tamaño de la `AbsoluteLayout` sí mismo. En otras palabras, el tamaño y la posición de cada niño es proporcional al tamaño del recipiente.

Curiosamente, esto es a menudo el caso con una Absoluto-

Diseño, y podría ser bueno si `AbsoluteLayout` alojadas estas situaciones de forma automática.

¡Lo hace!

`AbsoluteLayout` define una propiedad enlazable segundo adjunto, llamado `LayoutFlagsProperty`, y dos métodos más estáticas, nombrados `SetLayoutFlags` y `GetLayoutFlags`. Al establecer esta propiedad enlazable adjunto le permite especificar las coordenadas de posición del niño o tamaños (o ambos) que son proporcionales al tamaño de la `AbsoluteLayout`. Para la colocación de sus hijos, `AbsoluteLayout` escalas esas coordenadas y tamaños de forma adecuada.

Selecciona cómo esta característica funciona con uno o más miembros de la `AbsoluteLayoutFlags` enumeración:

- Ninguna (igual a 0)
- XProportional (1)
- YProportional (2)
- PositionProportional (3)
- WidthProportional (4)
- HeightProportional (8)
- SizeProportional (12)
- Todas (\xFFFFFFF)

Se puede establecer una posición proporcional y el tamaño de un niño de `AbsoluteLayout` utilizando los dos métodos estáticos:

```
AbsoluteLayout .SetLayoutBounds (vista, rect);  
AbsoluteLayout .SetLayoutFlags (vista, AbsoluteLayoutFlags .Todas);
```

O puede utilizar una versión de la `Añadir` método en el Niños colección que acepta una `AbsoluteLayoutFlags` miembro de la enumeración:

```
absoluteLayout.Children.Add (vista, rect, AbsoluteLayoutFlags .Todas);
```

Por ejemplo, si se utiliza el `SizeProportional` bandera y establecer el ancho del niño a 0,25 y la altura a 0,10, el niño será una cuarta parte de la anchura de la `AbsoluteLayout` y una décima parte de la altura. Suficientemente fácil.

los `PositionProportional` la bandera es similar, pero se necesita el tamaño del niño en cuenta: a cargo de (0, 0) pone al niño en la esquina superior izquierda, una posición de (1, 1) pone al niño en la esquina inferior derecha , y una posición de (0.5, 0.5) centra el niño dentro de la `AbsoluteLayout`. Tomando el tamaño del niño en cuenta es grande para algunas tareas, tales como centrar un niño en una `AbsoluteLayout` o mostrarla en contra de la derecha o inferior de canto, pero un poco incómoda para otras tareas.

A continuación se **ChessboardProportional**. La mayor parte del trabajo de posicionamiento y dimensionamiento se ha trasladado de nuevo al constructor. los `SizeChanged` manejador de ahora simplemente mantiene la relación general de aspecto mediante el establecimiento de la `WidthRequest` y `HeightRequest` propiedades de la `AbsoluteLayout` al mínimo de la anchura y la altura de la `ContentView`. Quitar ese `SizeChanged` la manipulación y el tablero de ajedrez se expande al tamaño de la página menos el relleno.

```
clase pública ChessboardProportionalPage : Pagina de contenido
{
    AbsoluteLayout AbsoluteLayout;

    público ChessboardProportionalPage ()
    {
        AbsoluteLayout = nuevo AbsoluteLayout
        {
            BackgroundColor = Color .FromRgb (240, 220, 130),
            HorizontalOptions = LayoutOptions .Centrar,
            VerticalOptions = LayoutOptions .Centrar
        };

        para ( En t fila = 0; fila <8; fila++)
        {
            para ( En t col = 0; Col <8; col++)
            {
                // Saltar cualquier otro cuadrado.
                Si (((Fila ^ col) y 1) == 0)
                    continuar ;

                BoxView boxView = nuevo BoxView
                {
                    color = Color .FromRgb (0, 64, 0)
                };

                Rectángulo rect = nuevo Rectángulo (Col / 7,0,
                                                fila / 7,0,
                                                1 / 8,0,
                                                1 / 8,0);

                absoluteLayout.Children.Add (boxView, rect, AbsoluteLayoutFlags .Todas);
            }
        }
    }
}
```

```

ContentView contentView = nuevo ContentView
{
    Content = AbsoluteLayout
};

contentView.SizeChanged += OnContentSizeChanged;

esta.Padding = nuevo Espesor (5, Dispositivo.OnPlatform (25, 5, 5), 5, 5);
esta.Este contenido ha sido = contentView;
}

vacío OnContentSizeChanged ( objeto remitente, EventArgs args)
{
    ContentView contentView = ( ContentView )remitente;
    doble boardSize = Mates .min (contentView.Width, contentView.Height);
    absoluteLayout.WidthRequest = boardSize;
    absoluteLayout.HeightRequest = boardSize;
}
}
}

```

La pantalla se ve igual que el **ChessboardDynamic** programa.

Cada BoxView se añade a la AbsoluteLayout con el siguiente código. Todos los denominadores son valores de punto flotante, por lo que los resultados de las divisiones se convierten a doble:

```

Rectángulo rect = nuevo Rectángulo (Col / 7,0,                                // X
                                    fila / 7,0,                                // y
                                    1 / 8,0,                                 // anchura
                                    1 / 8,0);                                // altura

absoluteLayout.Children.Add (boxView, rect, AbsoluteLayoutFlags .Todas);

```

La anchura y la altura son siempre igual a un octavo de la anchura y la altura de la AbsoluteLayout.

Eso está claro. Pero el fila y columna variables son divididas por 7 (en lugar de 8) para el relativo X y y coordina. los fila y columna variables en el para bucles varían de 0 a 7. El fila y columna

valores de 0 corresponden a izquierda o de la parte superior, pero fila y columna valores de 7 tendrán que asignar X y y coordenadas de 1 a la posición del niño contra el borde derecho o inferior.

Si usted piensa que podría necesitar algunas reglas sólidas para derivar las coordenadas proporcionales, sigue leyendo.

Trabajar con coordenadas proporcionales

Trabajar con posicionamiento proporcional en una AbsoluteLayout puede ser complicado. A veces es necesario para compensar el cálculo interno que tiene el tamaño en cuenta. Por ejemplo, es posible que prefiera para especificar las coordenadas de modo que un valor X de 1 significa que el borde izquierdo del niño se coloca en el borde derecho de la AbsoluteLayout, y que necesita para convertir eso a una coordenada que Absoluto-Diseño entiende.

En la discusión que sigue, una coordenada que hace *no* tomar en cuenta el tamaño, una coordenada en

que 1 significa que el niño se coloca justo fuera del borde derecho o inferior de la AbsoluteLayout—fuera—se refiere como una *fraccionario* coordinar. El objetivo de esta sección es desarrollar reglas para convertir una fracción de coordinar a un proporcional coordenada que se puede utilizar con AbsoluteLayout. Esta conversión requiere que se conoce el tamaño de la vista de los niños.

Supongamos que usted está poniendo una vista llamada *niño* en una AbsoluteLayout llamado *AbsoluteLayout*, con un rectángulo límites de diseño para el niño nombrado *layoutBounds*. Vamos a limitar este análisis a las coordenadas horizontales y tamaños. El proceso es el mismo para las coordenadas verticales y tamaños.

Este niño debe primero obtener un ancho de alguna manera. El niño podría calcular su propia anchura o una anchura en unidades independientes del dispositivo podría ser asignado a ella a través de la *LayoutParams* propiedad adjunta. Pero vamos a suponer que la *AbsoluteLayoutFlags.WidthProportional* indicador está activado, lo que significa que la anchura se calcula a partir de la Anchura ámbito de los límites de diseño y la anchura de la *AbsoluteLayout*.

fueras:

$$\text{relativeX} = \frac{\text{width} * \text{childWidth}}{\text{parentWidth}}$$

Si el *AbsoluteLayoutFlags.XProportional* bandera se fija también, entonces internamente la *AbsoluteLayout* fuera calcula una coordenada para el niño con relación a sí mismo, tomando el tamaño del niño en cuenta:

$$\text{relativeX} = (\text{width} - \text{childWidth}) * \text{width}$$

Por ejemplo, si el *AbsoluteLayout* tiene una anchura de 400, y el niño tiene una anchura de 100, y *layoutBounds.X* es 0,5, entonces *relativeChildCoordinate.X* se calcula como 150. Esto significa que el borde izquierdo del niño es 150 píxeles desde el borde izquierdo de la matriz. Eso hace que el niño se centre horizontalmente dentro de la *AbsoluteLayout*.

También es posible calcular coordenadas de un niño fraccionaria:

$$\text{relativeX} = \frac{\text{width} * \text{childWidth}}{\text{parentWidth}}$$

Este no es el mismo que el de coordenadas proporcionales porque un niño fraccional coordenada de 1 significa que el borde izquierdo del niño está justo fuera del borde derecho de la *AbsoluteLayout*, y por lo tanto el niño se encuentra fuera de la superficie de la *AbsoluteLayout*. Para continuar con el ejemplo, coordenadas es 150 dividido por 400 o 0.375 el niño fraccionada. La izquierda de la vista del niño se coloca en (0.375 * 400) o 150 unidades desde el borde izquierdo de la *AbsoluteLayout*.

Vamos a reorganizar los términos de la fórmula que calcula el niño coordenada relativa para resolver *layoutBounds.X*:

$$\text{relativeX} = \frac{\text{width} * (\text{width} - \text{childWidth})}{\text{width} * \text{childWidth}}$$

Y vamos a dividir la parte superior e inferior de esa relación en el ancho de la *AbsoluteLayout*:

$$\text{relativeX} = \frac{(1 - \frac{\text{childWidth}}{\text{width}}) * \text{width}}{(1 - \frac{\text{childWidth}}{\text{width}}) * \text{width}}$$

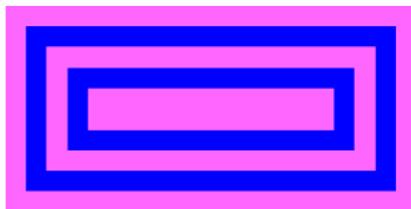
Si también está utilizando el ancho proporcional, entonces esa proporción en el denominador es diseño-Bounds.Width:

$$\text{ancho} = \frac{\text{anchura}}{(1 - \frac{\text{anchura}}{\text{alto}}) \cdot \text{alto}}$$

Y que a menudo es una fórmula muy útil, ya que permite convertir a un niño fraccional coordinar a un proporcional de coordenadas para su uso en el rectángulo límites de diseño.

En el **ChessboardProportional** ejemplo, cuando columna es igual a 7, la fractionalChildCoordinate.X se 7 dividido por el número de columnas (8), o 7/8. El denominador es 1 menos 1/8 (la anchura proporcional de la plaza), o 7/8 de nuevo. La relación es 1.

Veamos un ejemplo donde se aplica la fórmula de código para las coordenadas fraccionarias. los **ProportionalCoordinateCalc** programa intenta reproducir esta figura simple usando ocho azul BoxView los elementos de una rosa AbsoluteLayout:



Toda la figura tiene un 2: aspecto 1. Se puede pensar en la figura como que comprende cuatro rectángulos horizontales y cuatro rectángulos verticales. Los pares de rectángulos azules horizontales en la parte superior e inferior tienen una altura de 0,1 unidades fraccionarias (con relación a la altura de la AbsoluteLayout) y están espaciados 0,1 unidades de la parte superior e inferior y entre sí. Los rectángulos azules verticales parecen estar espaciados y de tamaño similar, pero debido a la relación de aspecto es de 2: 1, los rectángulos verticales tener una anchura de 0,05 unidades y están espaciados con 0,05 unidades de la izquierda y derecha y entre sí.

los AbsoluteLayout se define y centrado en un archivo XAML y de color rosa:

```
<Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns:x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " ProportionalCoordinateCalc.ProportionalCoordinateCalcPage " >

<ContentPage.Padding >
    <OnPlatform x: TypeArguments = " Espesor "
        iOS = " 5, 25, 5, 5 "
        Androide = " 5 "
        WinPhone = " 5 " />
</ContentPage.Padding >

<ContentView SizeChanged = " OnContentViewSizeChanged " >
    <AbsoluteLayout x: Nombre = " AbsoluteLayout "
        Color de fondo = " Rosado "
        HorizontalOptions = " Centrar "
        VerticalOptions = " Centrar " />
</ContentView >
```

</ Pagina de contenido >

El archivo de código subyacente se define una matriz de Rectángulo estructuras con las coordenadas fraccionales para cada uno de los ocho BoxView elementos. en un para cada bucle, el programa aplica una ligera variación de la fórmula final se muestra arriba. En lugar de un denominador igual a 1 menos el valor de diseño-

Bounds.Width (o layoutBounds.Height), que utiliza el Anchura (o Altura) de los límites fraccionarios, que es el mismo valor.

```
público clase parcial ProportionalCoordinateCalcPage : Pagina de contenido
{
    público ProportionalCoordinateCalcPage ()
    {
        InitializeComponent ();

        Rectángulo [] FractionalRects =
        {
            nuevo Rectángulo (0,05, 0,1, 0,90, 0,1), // superior exterior
            nuevo Rectángulo (0,05, 0,8, 0,90, 0,1), // inferior exterior
            nuevo Rectángulo (0,05, 0,1, 0,05, 0,8), // izquierda exterior
            nuevo Rectángulo (0,90, 0,1, 0,05, 0,8), // externa derecha

            nuevo Rectángulo (0,15, 0,3, 0,70, 0,1), // superior interior
            nuevo Rectángulo (0,15, 0,6, 0,70, 0,1), // inferior interna
            nuevo Rectángulo (0,15, 0,3, 0,05, 0,4), // izquierda interior
            nuevo Rectángulo (0,80, 0,3, 0,05, 0,4), // derecha interior
        };
    }

    para cada ( Rectángulo fractionalRect en FractionalRects )
    {
        Rectángulo layoutBounds = nuevo Rectángulo
        {
            // proporcionar cálculos de coordenadas.
            X = fractionalRect.X / ( 1 - fractionalRect.Width ),
            Y = fractionalRect.Y / ( 1 - fractionalRect.Height ),

            Anchura = fractionalRect.Width,
            Altura = fractionalRect.Height
        };

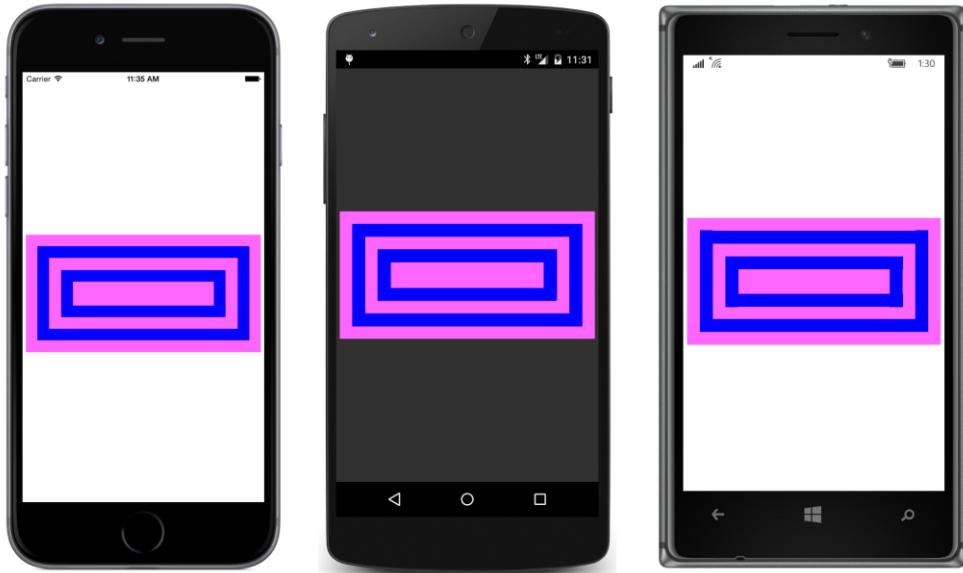
        absoluteLayout.Children.Add (
            nuevo BoxView
            {
                color = Color .Azul
            },
            layoutBounds,
            AbsoluteLayoutFlags .Todas);
    }
}

vacío OnContentViewSizeChanged ( objeto remitente, EventArgs args )
{
    ContentView contentView = ( ContentView )remitente;
```

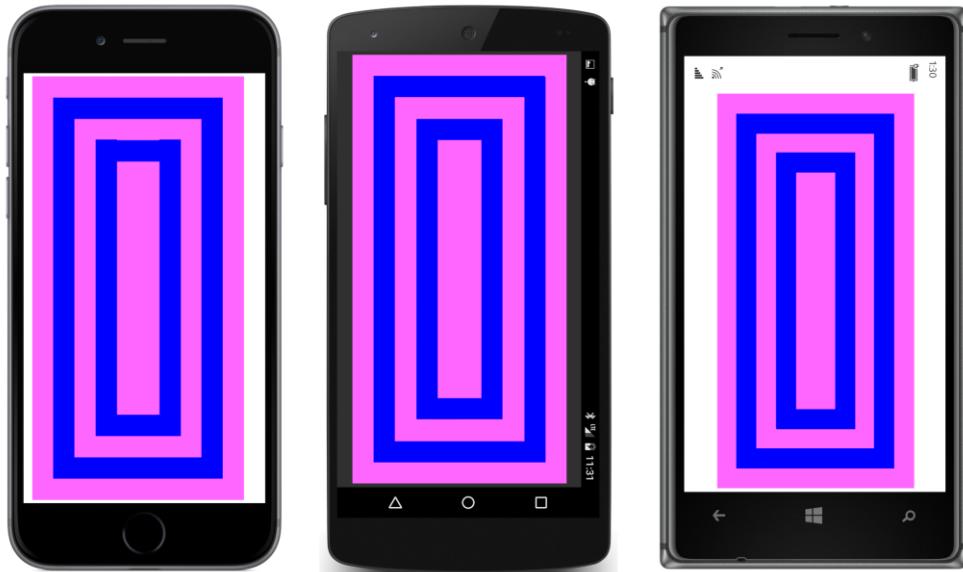
```
// figura tiene una relación de aspecto de 2: 1.  
double height = Math.min(contentView.Width / 2, contentView.Height);  
absoluteLayout.WidthRequest = 2 * altura;  
absoluteLayout.HeightRequest = altura;  
}  
}
```

los SizeChanged manejador simplemente fija la relación de aspecto.

Aquí está el resultado:



Y, por supuesto, se puede girar el teléfono hacia un lado y ver una figura más grande en el modo de paisaje, que usted tiene que ver girando de lado este libro:



AbsoluteLayout y XAML

Como hemos visto, se puede posicionar y tamaño de un niño de una `AbsoluteLayout` en código mediante el uso de una de las

Añadir métodos disponibles en la Niños recolección o mediante el establecimiento de una propiedad unido a través de una llamada al método estático.

Pero, ¿cómo diablos se establece la posición y el tamaño de `AbsoluteLayout` niños en XAML?

Una sintaxis muy especial está involucrado. Esta sintaxis se ilustra por esta versión XAML de la anterior **AbsoluteDemo** programa, llamado **AbsoluteXamlDemo**:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " AbsoluteXamlDemo.AbsoluteXamlDemoPage " >

    < AbsoluteLayout Relleno = " 50 " >
        < BoxView Color = " Acento "
            AbsoluteLayout.LayoutBounds = " 0, 10, 200, 5 " />

        < BoxView Color = " Acento "
            AbsoluteLayout.LayoutBounds = " 0, 20, 200, 5 " />

        < BoxView Color = " Acento "
            AbsoluteLayout.LayoutBounds = " 10, 0, 5, 65 " />

        < BoxView Color = " Acento "
            AbsoluteLayout.LayoutBounds = " 20, 0, 5, 65 " />
```

```

< Etiqueta Texto = "Cabecera con estilo "
    Tamaño de fuente = "24 "
    AbsoluteLayout.LayoutBounds = "30, 25, AutoSize, AutoSize " />

< Etiqueta AbsoluteLayout.LayoutBounds = "0, 80, AutoSize, AutoSize " >
    < Label.FormattedText >
        < FormattedString >
            < Largo Texto = "A pesar de que " />
            < Largo Texto = "AbsoluteLayout "
                FontAttributes = "Itálico " />
            < Largo Texto =
                " por lo general se emplea para otros propósitos
                que la visualización de texto usando " />
                < Largo Texto = " Etiqueta "
                    FontAttributes = "Itálico " />
                < Largo Texto =
                    " , Es obvio que se puede utilizar de esa manera.
                    El texto continúa para envolver muy bien
                    dentro de los límites del contenedor
                    y cualquier relleno que podrían ser aplicados. " />
                    </ FormattedString >
                </ Label.FormattedText >
            </ Largo >
        </ AbsoluteLayout >
    </ Pagina de contenido >

```

El archivo de código subyacente contiene sólo una InitializeComponent llamada.

Aquí está la primera BoxView:

```

< BoxView Color = "Acento "
    AbsoluteLayout.LayoutBounds = "0, 10, 200, 5 " />

```

En XAML, una propiedad enlazable adjunta se expresa como un atributo que consiste en un nombre de clase (`AbsoluteLayout`) y un nombre de propiedad (`LayoutBounds`) separados por un punto. Cada vez que vea un atributo tal, es siempre una propiedad que puede vincularse adjunto. Esa es la única aplicación de esta sintaxis de atributo.

En resumen, las combinaciones de nombres de clases y nombres de propiedades sólo aparecen en XAML en tres contextos específicos: Si aparecen como elementos, que son elementos de la propiedad. Si aparecen como atributos, propiedades enlazables están unidos. Y el único otro contexto para un nombre de clase y la propiedad de nombre es un argumento a una `x: Estático extensión de marcado`.

los `AbsoluteLayout.LayoutBounds` atributo se establece comúnmente a cuatro números separados por comas. También puede expresar `AbsoluteLayout.LayoutBounds` como un elemento de propiedad:

```

< BoxView Color = "Acento " >
    < AbsoluteLayout.LayoutBounds >
        0, 10, 200, 5
    </ AbsoluteLayout.LayoutBounds >
</ BoxView >

```

Esos cuatro números son analizados por el BoundsTypeConverter y no el RectangleTypeConverter porque el BoundsTypeConverter permite el uso de Tamaño automático para las partes de anchura y altura. Puedes ver el Tamaño automático argumentos más adelante en el **AbsoluteXamlDemo** archivo XAML:

```
< Etiqueta Texto = " Cabecera con estilo "
    Tamaño de fuente = " 24 "
    AbsoluteLayout.LayoutBounds = " 30, 25, AutoSize, AutoSize " />
```

O bien, puede dejarlos fuera:

```
< Etiqueta Texto = " Cabecera con estilo "
    Tamaño de fuente = " 24 "
    AbsoluteLayout.LayoutBounds = " 30, 25 " />
```

Lo curioso de propiedades enlazables adjuntos que se especifica en XAML es que en realidad no existen! No hay ningún campo, propiedad o método en el **AbsoluteLayout** llamado **LayoutBounds**. Ciertamente, hay una public static campo de sólo lectura de tipo **BindableProperty** llamado **LayoutBoundsProperty**, y hay métodos estáticos públicos nombrados **SetLayoutBounds** y **GetLayoutBounds**. Pero no hay nada nombrado **LayoutBounds**. El analizador XAML reconoce la sintaxis como una referencia a una propiedad enlazable adjunto y después busca **LayoutBoundsProperty** en el **AbsoluteLayout** clase. Desde allí se puede llamar Valor ajustado en la vista de destino con esa **BindableProperty** objeto junto con el valor de la **BoundsTypeConverter**.

los **Tablero de ajedrez** serie de programas parece un candidato poco probable para la duplicación en XAML porque el archivo necesitaría 32 casos de **BoxView** sin el beneficio de bucles. sin embargo, el **ChessboardXaml** programa muestra cómo especificar dos propiedades de **BoxView** en un estilo implícita, incluyendo el **AbsoluteLayout.LayoutStyle** adjunta propiedad enlazable:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " ChessboardXaml.ChessboardXamlPage " >

    < ContentPage.Padding >
        < OnPlatform x: TypeArguments = " Espesor "
            iOS = " 5, 25, 5, 5 "
            Androide = " 5 "
            WinPhone = " 5 " />
    </ ContentPage.Padding >

    < ContentPage.Resources >
        < ResourceDictionary >
            < Estilo Tipo de objetivo = " BoxView " >
                < Setter Propiedad = " Color " Valor = "# 004000 " />
                < Setter Propiedad = " AbsoluteLayout.LayoutStyle " Valor = " Todas " />
            </ Estilo >
        </ ResourceDictionary >
    </ ContentPage.Resources >

    < ContentView SizeChanged = " OnContentViewSizeChanged " >
        < AbsoluteLayout x: Nombre = " AbsoluteLayout "
            Color de fondo = "# F0DC82 " />
```

```

VerticalOptions = "Centrar"
HorizontalOptions = "Centrar" >

< BoxView AbsoluteLayout.LayoutBounds = "0,0,0,0,0,125,0,125" />
< BoxView AbsoluteLayout.LayoutBounds = "0,29,0,0,0,125,0,125" />
< BoxView AbsoluteLayout.LayoutBounds = "0,57,0,0,0,125,0,125" />
< BoxView AbsoluteLayout.LayoutBounds = "0,86,0,0,0,125,0,125" />

< BoxView AbsoluteLayout.LayoutBounds = "0,14,0,14,0,125,0,125" />
< BoxView AbsoluteLayout.LayoutBounds = "0,43,0,14,0,125,0,125" />
< BoxView AbsoluteLayout.LayoutBounds = "0,71,0,14,0,125,0,125" />
< BoxView AbsoluteLayout.LayoutBounds = "1,00,0,14,0,125,0,125" />

< BoxView AbsoluteLayout.LayoutBounds = "0,0,0,29,0,125,0,125" />
< BoxView AbsoluteLayout.LayoutBounds = "0,29,0,29,0,125,0,125" />
< BoxView AbsoluteLayout.LayoutBounds = "0,57,0,29,0,125,0,125" />
< BoxView AbsoluteLayout.LayoutBounds = "0,86,0,29,0,125,0,125" />

< BoxView AbsoluteLayout.LayoutBounds = "0,14,0,43,0,125,0,125" />
< BoxView AbsoluteLayout.LayoutBounds = "0,43,0,43,0,125,0,125" />
< BoxView AbsoluteLayout.LayoutBounds = "0,71,0,43,0,125,0,125" />
< BoxView AbsoluteLayout.LayoutBounds = "1,00,0,43,0,125,0,125" />

< BoxView AbsoluteLayout.LayoutBounds = "0,0,0,57,0,125,0,125" />
< BoxView AbsoluteLayout.LayoutBounds = "0,29,0,57,0,125,0,125" />
< BoxView AbsoluteLayout.LayoutBounds = "0,57,0,57,0,125,0,125" />
< BoxView AbsoluteLayout.LayoutBounds = "0,86,0,57,0,125,0,125" />

< BoxView AbsoluteLayout.LayoutBounds = "0,14,0,71,0,125,0,125" />
< BoxView AbsoluteLayout.LayoutBounds = "0,43,0,71,0,125,0,125" />
< BoxView AbsoluteLayout.LayoutBounds = "0,71,0,71,0,125,0,125" />
< BoxView AbsoluteLayout.LayoutBounds = "1,00,0,71,0,125,0,125" />

< BoxView AbsoluteLayout.LayoutBounds = "0,0,0,86,0,125,0,125" />
< BoxView AbsoluteLayout.LayoutBounds = "0,29,0,86,0,125,0,125" />
< BoxView AbsoluteLayout.LayoutBounds = "0,57,0,86,0,125,0,125" />
< BoxView AbsoluteLayout.LayoutBounds = "0,86,0,86,0,125,0,125" />

< BoxView AbsoluteLayout.LayoutBounds = "0,14,1,00,0,125,0,125" />
< BoxView AbsoluteLayout.LayoutBounds = "0,43,1,00,0,125,0,125" />
< BoxView AbsoluteLayout.LayoutBounds = "0,71,1,00,0,125,0,125" />
< BoxView AbsoluteLayout.LayoutBounds = "1,00,1,00,0,125,0,125" />

</ AbsoluteLayout >
</ ContentView >
</ Pagina de contenido >
```

Sí, es una gran cantidad de persona BoxView elementos, pero no se puede discutir con la limpieza del archivo. El archivo de código subyacente simplemente ajusta la relación de aspecto:

```

pública clase parcial ChessboardXamlPage : Pagina de contenido
{
    pública ChessboardXamlPage ()
    {
        InitializeComponent ();
    }
}
```

```

        }

        vacío OnContentViewSizeChanged ( objeto remitente, EventArgs args)
        {
            ContentView contentView = ( ContentView )remitente;
            doble boardSize = Mates .min (contentView.Width, contentView.Height);
            absoluteLayout.WidthRequest = boardSize;
            absoluteLayout.HeightRequest = boardSize;
        }
    }
}

```

superposiciones

La capacidad de solapar los niños en el AbsoluteLayout tiene algunas aplicaciones interesantes y útiles, entre ellas la capacidad de cubrir toda la interfaz de usuario con algo a veces llamada *cubrir*. Tal vez su página está llevando a cabo un trabajo largo y que no quiere que el usuario interactúa con la página hasta que se complete el trabajo. Puede colocar una superposición semitransparente sobre la página y tal vez mostrar una ActivityIndicator o una Barra de progreso.

Aquí hay un programa llamado **SimpleOverlay** que muestra esta técnica. El archivo XAML comienza con una AbsoluteLayout ocupando toda la página. El primer hijo de ese AbsoluteLayout es un Apilar-Diseño, que desea llenar la página también. Sin embargo, el valor predeterminado HorizontalOptions y verticalOptions ajustes de Llenar sobre el StackLayout no funcionan para los niños de una AbsoluteLayout. En cambio, el StackLayout llena el AbsoluteLayout mediante el uso de la AbsoluteLayout.LayoutBounds y AbsoluteLayout.LayoutFlags adjunta propiedades enlazables:

```

< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " SimpleOverlay.SimpleOverlayPage " >

    < AbsoluteLayout >
        < StackLayout AbsoluteLayout.LayoutBounds = " 0, 0, 1, 1 "
                      AbsoluteLayout.LayoutFlags = " Todas " >
            < Etiqueta Texto =
                " Esto podría ser una página completa de objetos de interfaz de usuario, excepto
                que la única funcional objeto de interfaz de usuario en la página
                Es un botón. "
                Tamaño de fuente = " Medio "
                VerticalOptions = " CenterAndExpand "
                HorizontalTextAlignment = " Centrar " />
            < Botón Texto = " Ejecutar 5-segundo empleo "
                Tamaño de fuente = " Grande "
                VerticalOptions = " CenterAndExpand "
                HorizontalOptions = " Centrar "
                hecho clic = " OnButtonClicked " />
            < Botón Texto = " Un botón que no hace nada "
                Tamaño de fuente = " Grande "
                VerticalOptions = " CenterAndExpand "

```

```

    HorizontalOptions = "Centrar" />

    < Etiqueta Texto =
    " Esto continúa la página llena de objetos de interfaz de usuario, excepto
    que la única funcional objeto de interfaz de usuario en la página
    es el botón. "
        Tamaño de fuente = "Medio"
        VerticalOptions = "CenterAndExpand"
        HorizontalTextAlignment = "Centrar" />
    </ StackLayout >

    <!-- Cubrir -->
    < ContentView x: Nombre = "cubrir"
        AbsoluteLayout.LayoutBounds = "0, 0, 1, 1"
        AbsoluteLayout.LayoutFlags = "Todas"
        Es visible = "Falso"
        Color de fondo = "#C0B080B0"
        Relleno = "10, 0" >

    < Barra de progreso x: Nombre = "barra de progreso"
        VerticalOptions = "Centrar" />

    </ ContentView >
    </ AbsoluteLayout >
</ Página de contenido >

```

El segundo hijo de la AbsoluteLayout es un ContentView, que también llena el AbsoluteLayout

y, básicamente, se encuentra en la parte superior de la StackLayout. Sin embargo, observe que la Es visible propiedad se establece en Falso, lo que significa que este ContentView y sus hijos no participen en el diseño. Los ContentView es todavía un niño de la AbsoluteLayout, sino que simplemente ha saltado cuando el sistema de diseño es el encolado y la prestación de todos los elementos de la página.

Esta ContentView es la superposición. Cuando Es visible se establece en Cíerto, que bloquea la entrada del usuario a los puntos de vista por debajo de ella. Los Color de fondo se ajusta a un gris semitransparente, y una Barra de progreso se centra verticalmente dentro de ella.

UN Barra de progreso se asemeja a una deslizador sin pulgar. UN Barra de progreso siempre está orientada horizontalmente. No establezca la HorizontalOptions propiedad de una Barra de progreso a Inicio, Centro, o Fin a menos que también establece su WidthRequest propiedad.

Un programa puede indicar el progreso mediante el establecimiento de la Progreso propiedad de la Barra de progreso a un valor entre 0 y 1. Esto se demuestra en el hecho clic manejador para el único funcional Botón en el programa. Este controlador simula un trabajo muy largo que se realiza en el código con un temporizador que determina cuándo han transcurrido cinco segundos:

```

pública clase parcial SimpleOverlayPage : Página de contenido
{
    pública SimpleOverlayPage ()
    {
        InitializeComponent ();
    }
}

```

```
vacio OnButtonClicked ( objeto remitente, EventArgs args)
{
    // Muestra superposición con ProgressBar.
    overlay.isVisible = cierto;

    Espacio de tiempo duración = Espacio de tiempo .FromSeconds (5);
    Fecha y hora horaInicio = Fecha y hora .Ahora;

    // Iniciar temporizador.
    Dispositivo .StartTimer ( Espacio de tiempo .FromSeconds (0,1), () =>
    {
        doble progreso = ( Fecha y hora .now - fecha de inicio) .TotalMilliseconds /
            duration.TotalMilliseconds;

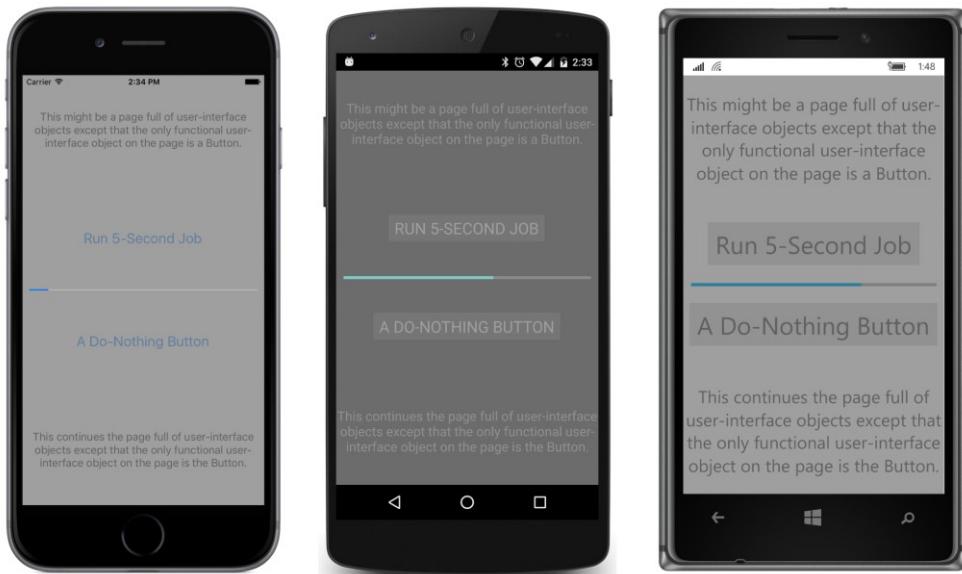
        ProgressBar.progress = progreso;
        bool continueTimer = progreso <1;

        Si ( ! ContinueTimer)
        {
            // Ocultar una suposición.
            overlay.isVisible = falso ;
        }
        regreso continueTimer;
    });
}
}
```

los hecho clic manejador comienza estableciendo la Es visible propiedad de la superposición de cierto, que revela la superposición y su hijo Barra de progreso y evita aún más la interacción con la interfaz de usuario debajo. El temporizador se establece para un décimo segundo y calcula un nuevo Progreso propiedad para el

Barra de progreso en función del tiempo transcurrido. Cuando los cinco segundos han aumentado, la superposición se oculta de nuevo y los retornos temporizador de devolución de llamada falso.

Esto es lo que parece que con la superposición que cubren la página y el largo trabajo en curso:



Una superposición no tiene por qué limitarse a una Barra de progreso o un ActivityIndicator. Puede incluir una Cancelar botón o otros puntos de vista.

Un poco de diversión

Como podrán ver por ahora, la AbsoluteLayout se utiliza a menudo para algunos propósitos especiales que no serían fáciles de otra manera.

Algunas de estas, en realidad podría ser clasificado como "divertido".

DotMatrixClock muestra los dígitos de la hora actual utilizando una pantalla de simulación de 5×7 de matriz de puntos. Cada punto es una BoxView, el tamaño y posición en la pantalla y de color de forma individual, ya sea rojo o gris claro, dependiendo de si el punto está dentro o fuera. Es concebible que los puntos de este reloj podrían organizarse en anidada StackLayout elementos o una Cuadrícula, pero cada uno BoxView Hay que dar un tamaño de todos modos. La cantidad absoluta y la regularidad de estos puntos de vista sugiere que el programador sabe mejor que una clase de diseño de la forma de organizar en la pantalla, porque StackLayout y Cuadrícula tenga que realizar los cálculos de localización de una manera más generalizada. Por esa razón, este es un trabajo ideal para AbsoluteLayout.

fuerá.

Un archivo XAML pone un poco de relleno en la página y prepara una AbsoluteLayout para el llenado por código:

```
<Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns:x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " DotMatrixClock.DotMatrixClockPage "
    Relleno = " 10 "
    SizeChanged = " OnPageSizeChanged " >

    <AbsoluteLayout x: Nombre = " AbsoluteLayout "
```

```

VerticalOptions = "Centrar" />

</ Pagina de contenido >

El archivo de código subyacente contiene varios campos, entre ellos dos matrices, nombrados numberPatterns y colonPattern, que definen los patrones de matriz de puntos para los 10 dígitos y un separador de colon:

público clase parcial DotMatrixClockPage : Pagina de contenido
{
    // Total puntos horizontal y verticalmente.
    const int horzDots = 41;
    const int vertDots = 7;

    // 5 x 7 puntos patrones de matriz de 0 a 9.
    estático int sólo lectura[,] numberPatterns = new int [10,7,5]
    {
        {
            {
                {0, 1, 1, 1, 0}, {1, 0, 0, 0, 1}, {1, 0, 0, 1, 1}, {1, 0, 1, 0, 1},
                {1, 1, 0, 0, 1}, {1, 0, 0, 0, 1}, {0, 1, 1, 1, 0}
            },
            {
                {0, 0, 1, 0, 0}, {0, 1, 1, 0, 0}, {0, 0, 1, 0, 0}, {0, 0, 1, 0, 0},
                {0, 0, 1, 0, 0}, {0, 0, 1, 0, 0}, {0, 1, 1, 1, 0}
            },
            {
                {0, 1, 1, 1, 0}, {1, 0, 0, 0, 1}, {0, 0, 0, 0, 1}, {0, 0, 0, 1, 0},
                {0, 0, 1, 0, 0}, {0, 1, 0, 0, 0}, {1, 1, 1, 1, 1}
            },
            {
                {1, 1, 1, 1, 1}, {0, 0, 0, 1, 0}, {0, 0, 1, 0, 0}, {0, 0, 0, 1, 0},
                {0, 0, 0, 0, 1}, {1, 0, 0, 0, 1}, {0, 1, 1, 1, 0}
            },
            {
                {0, 0, 0, 1, 0}, {0, 0, 1, 1, 0}, {0, 1, 0, 1, 0}, {1, 0, 0, 1, 0},
                {1, 1, 1, 1, 1}, {0, 0, 0, 1, 0}, {0, 0, 0, 1, 0}
            },
            {
                {1, 1, 1, 1, 1}, {1, 0, 0, 0, 0}, {1, 1, 1, 1, 0}, {0, 0, 0, 0, 1},
                {0, 0, 0, 0, 1}, {1, 0, 0, 0, 1}, {0, 1, 1, 1, 0}
            },
            {
                {0, 0, 1, 1, 0}, {0, 1, 0, 0, 0}, {1, 0, 0, 0, 0}, {1, 1, 1, 1, 0},
                {1, 0, 0, 0, 1}, {1, 0, 0, 0, 1}, {0, 1, 1, 1, 0}
            },
            {
                {1, 1, 1, 1, 1}, {0, 0, 0, 0, 1}, {0, 0, 0, 1, 0}, {0, 0, 1, 0, 0},
                {0, 1, 0, 0, 0}, {0, 1, 0, 0, 0}, {0, 1, 0, 0, 0}
            },
            {
                {0, 1, 1, 1, 0}, {1, 0, 0, 0, 1}, {1, 0, 0, 0, 1}, {0, 1, 1, 1, 0},
                {1, 0, 0, 0, 1}, {1, 0, 0, 0, 1}, {0, 1, 1, 1, 0}
            },
            {
                {0, 1, 1, 1, 0}, {1, 0, 0, 0, 1}, {1, 0, 0, 0, 1}, {0, 1, 1, 1, 1},
                {0, 1, 1, 1, 0}, {1, 0, 0, 0, 1}, {1, 0, 0, 0, 1}
            }
        }
    }
}

```

```

        {0, 0, 0, 0, 1}, {0, 0, 0, 1, 0}, {0, 1, 1, 0, 0}
    },
};

// patrón de matriz de puntos para un colon.
estático int sólo lectura[,] ColonPattern = new int[7, 2]
{
    {0, 0}, {1, 1}, {1, 1}, {0, 0}, {1, 1}, {1, 1}, {0, 0}
};

// BoxView colores para encendido y apagado.
estático solo lectura Color Colorm = Color.Rojo;
estático solo lectura Color COLOROFF = nuevo Color(0.5, 0.5, 0.5, 0.25);

// vistas de la caja para 6 dígitos, 7 filas, 5 columnas.
BoxView[,] digitBoxViews = nuevo BoxView[6, 7, 5];
...
}

```

Los campos también se definen por una serie de BoxView objetos para los seis dígitos de los de tiempo de dos dígitos cada uno para la hora, minutos y segundos. El número total de puntos horizontal (establecer como horzDots) incluye cinco puntos para cada uno de los seis dígitos, cuatro puntos para el colon entre la hora y los minutos, cuatro para el colon entre los minutos y segundos, y una anchura en un solo punto entre los dígitos de otra manera.

constructor del programa (que se muestra a continuación) crea un total de 238 BoxView Los objetos y los añade a una AbsoluteLayout, sino que también ahorra el BoxView objetos para los dígitos en el digitBoxViews formación. (En teoría, el BoxView los objetos pueden ser referenciados más adelante mediante la indexación de la Niños colección de la AbsoluteLayout. Pero en esa colección, aparecen simplemente como una lista lineal. El almacenamiento también en una matriz multidimensional permite que sean identificados y referenciados más fácilmente.) Todo el posicionamiento y dimensionamiento se basa proporcional en una AbsoluteLayout que se supone que tiene una relación de aspecto de 41 a 7, que comprende el 41 BoxView anchuras y 7 BoxView alturas.

```

público clase parcial DotMatrixClockPage : Pagina de contenido
{
    ...
    público DotMatrixClockPage ()
    {
        InitializeComponent ();

        // BoxView dimensiones de puntos.
        doble altura = 0.85 / vertDots;
        doble anchura = 0.85 / horzDots;

        // Crear y montar el BoxViews.
        doble xIncrement = 1.0 / (horzDots - 1);
        doble yIncrement = 1.0 / (vertDots - 1);
        doble x = 0;

        para (En t dígito = 0; dígitos <6; dígitos++)
        {
            para (En t col = 0; Col <5; col++)
            {

```

```

doble y = 0;

para ( En t fila = 0; fila <7; fila++)
{
    // crear el digito BoxView y añadir al diseño.
    BoxView boxView = nuevo BoxView ();
    digitBoxViews [dígitos, fila, columna] = boxView;
    absoluteLayout.Children.Add (boxView,
        nuevo Rectángulo (X, y, anchura, altura),
        AbsoluteLayoutFlags .Todas);

    y += yIncrement;
}
x += xIncrement;
}

x += xIncrement;

// Los dos puntos entre las horas, minutos y segundos.
Si (== 1 || == dígitos dígitos 3)
{
    En t de colon = dígitos / 2;

    para ( En t col = 0; Col <2; col++)
    {
        doble y = 0;

        para ( En t fila = 0; fila <7; fila++)
        {
            // Crear el BoxView y establecer el color.
            BoxView boxView = nuevo BoxView
            {
                Color = colonPattern [fila, columna] == 1?
                    Coloron: COLOROFF
                };
            absoluteLayout.Children.Add (boxView,
                nuevo Rectángulo (X, y, anchura, altura),
                AbsoluteLayoutFlags .Todas);

            y += yIncrement;
        }
        x += xIncrement;
    }
    x += xIncrement;
}

// Establecer el temporizador e inicializar con una llamada manual.
Dispositivo .StartTimer ( Espacio de tiempo .FromSeconds (1), OnTimer);
A tiempo();
}

...
}

```

Como se recordará, el horzDots y vertDots constantes se establecen en 41 y 7, respectivamente. Para llenar el AbsoluteLayout, cada BoxView necesita ocupar una fracción de la anchura igual a 1 / horzDots

y una fracción de la altura igual a $1 / \text{vertDots}$. La altura y la anchura se establece en cada BoxView es del 85 por ciento de ese valor para separar los puntos suficientes para que no se ejecutan en uno a:

```
doble altura = 0.85 / vertDots;
doble anchura = 0.85 / horzDots;
```

Para posicionar cada BoxView, el constructor calcula proporcional xincrement y yincrement

Los valores de este modo:

```
doble xincrement = 1,0 / (horzDots - 1);
doble yincrement = 1,0 / (vertDots - 1);
```

Los denominadores aquí son 40 y 6 de manera que la X final y coordenadas de posición Y son valores de 1.

los BoxView objetos para los dígitos de la hora no se colorean en absoluto en el constructor, pero las de los dos puntos dobles se les da una Color propiedad basada en el colonPattern formación. los DotMatrixClockPage constructor concluye por un temporizador de un segundo.

los SizeChanged manejador de la página se establece desde el archivo XAML. los AbsoluteLayout se estira automáticamente horizontalmente para llenar la anchura de la página (menos el relleno), por lo que la HeightRe- búsqueda en realidad sólo establece la relación de aspecto:

```
público clase parcial DotMatrixClockPage : Pagina de contenido
{
    ...
    vacío OnPageSizeChanged ( objeto remitente, EventArgs args)
    {
        // No hay posibilidad de una pantalla tendrá una relación de aspecto> 41: 7
        absoluteLayout.HeightRequest = vertDots * Ancho / horzDots;
    }
    ...
}
```

Parece que la Device.StartTimer controlador de eventos debe ser bastante complejo, ya que es responsable de establecer la Color propiedad de cada BoxView basado en los dígitos de la hora actual. Sin embargo, la similitud entre las definiciones de la numberPatterns matriz y el digitBox-

Puntos de vista array hace sorprendentemente sencillo:

```
público clase parcial DotMatrixClockPage : Pagina de contenido
{
    ...
    bool A tiempo()
    {
        Fecha y hora fechaHora = Fecha y hora .Ahora;

        // Convertir reloj de 24 horas de reloj de 12 horas.
        En t hora = (dateTime.Hour + 11)% 12 + 1;

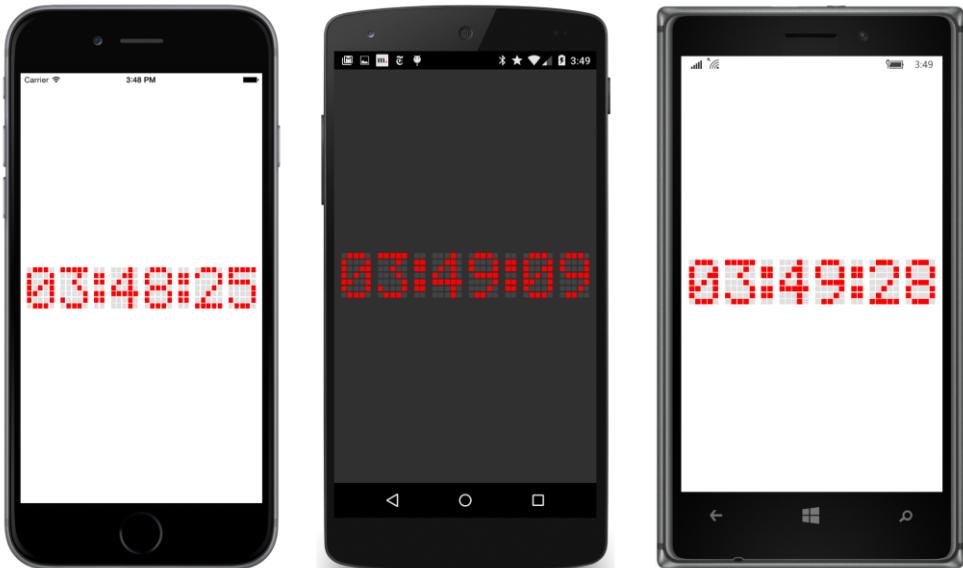
        // Establecer los colores de punto para cada dígito por separado.
        SetDotMatrix (0, horas / 10);
        SetDotMatrix (1, horas% 10);
    }
}
```

```
SetDotMatrix (2, dateTime.Minute / 10);
SetDotMatrix (3, dateTime.Minute% 10);
SetDotMatrix (4, dateTime.Second / 10);
SetDotMatrix (5, dateTime.Second% 10);

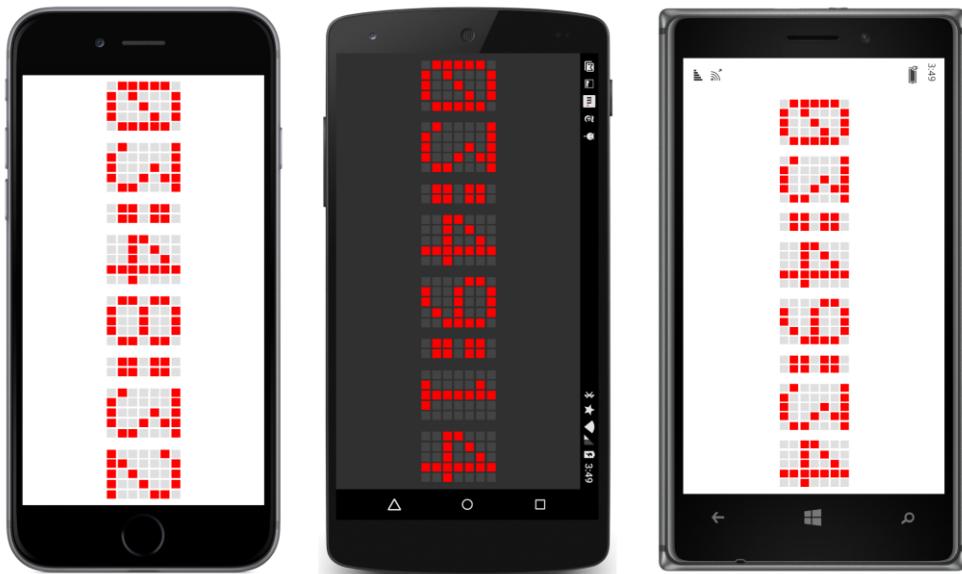
return true ;
}

vacío SetDotMatrix ( Ent Indice, Ent dígito)
{
    para ( Ent fila = 0; fila <7; fila++)
        para ( Ent col = 0; Col <5; col++)
    {
        bool lson = numberPatterns [dígitos, fila, columna] == 1;
        Color color = lson? Coloron: COLOROFF;
        digitBoxViews [índice, fila, columna] .Color = color;
    }
}
}
```

Y aquí está el resultado:



Por supuesto, cuanto más grande mejor, por lo que es probable que desee para encender el teléfono (o libro) de lado por algo lo suficientemente grande como para leer desde el otro lado de la habitación:



Otro tipo especial de aplicación adecuada para AbsoluteLayout es la animación. los **BouncingText**

programa utilice su archivo XAML para crear una instancia de dos Etiqueta elementos:

```
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns:x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " BouncingText.BouncingTextPage " >

    < AbsoluteLayout >
        < Etiqueta x: Nombre = " label1 "
            Texto = " REBOTAR "
            Tamaño de fuente = " Grande "
            AbsoluteLayout.LayoutFlags = " PositionProportional " />

        < Etiqueta x: Nombre = " label2 "
            Texto = " REBOTAR "
            Tamaño de fuente = " Grande "
            AbsoluteLayout.LayoutFlags = " PositionProportional " />

    </ AbsoluteLayout >
</ Pagina de contenido >
```

Observe que el **AbsoluteLayout.LayoutFlags** atributos se establecen en **PositionProportional**. los

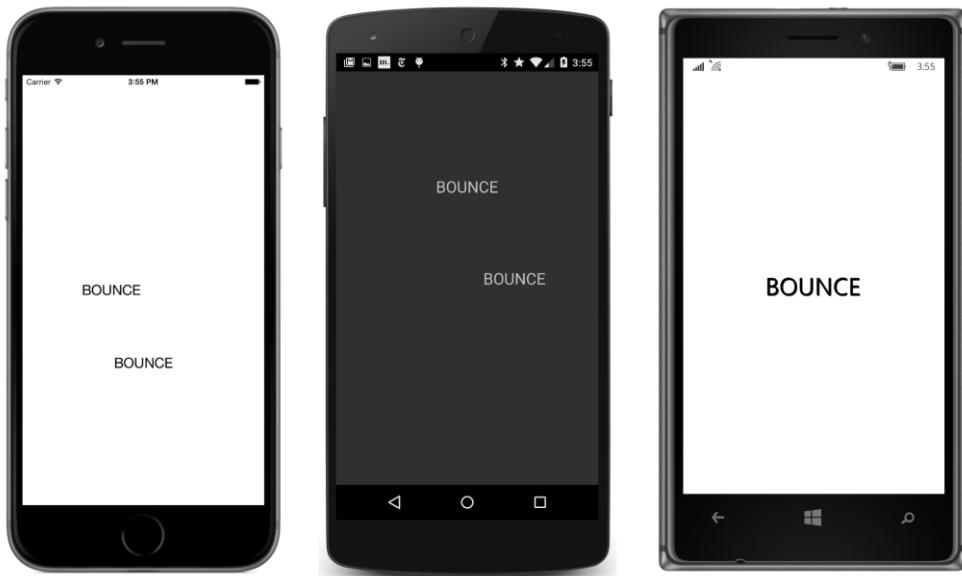
Etiqueta calcula su propio tamaño, pero el posicionamiento es proporcional. Los valores entre 0 y 1 pueden posicionar los dos Etiqueta elementos en cualquier lugar dentro de la página.

El archivo de código subyacente inicia un temporizador de ir con una duración de 15 milisegundos. Esto es equivalente a aproximadamente 60 pulsaciones por segundo, que es generalmente la frecuencia de actualización de las pantallas de video. Un temporizador de duración de 15 milisegundos es ideal para la realización de animaciones:

```
público clase parcial BouncingTextPage : Pagina de contenido
```

```
{  
    const doble período = 2.000; // en milisegundos  
    solo lectura Fecha y hora horaInicio = Fecha y hora .Ahora;  
  
    público BouncingTextPage ()  
    {  
        InitializeComponent ();  
        Dispositivo .StartTimer ( Espacio de tiempo .FromMilliseconds (15), OnTimerTick);  
    }  
  
    bool OnTimerTick ()  
    {  
        Espacio de tiempo transcurrido = Fecha y hora .now - fecha de inicio;  
        doble t = (elapsed.TotalMilliseconds período%) / período; // 0 a 1  
        t = 2 * (t < 0.5 t: 1 - t?); // 0 a 1 a 0  
  
        AbsoluteLayout .SetLayoutBounds (label1,  
            nuevo Rectángulo (T, 0.5, AbsoluteLayout .Tamaño automático, AbsoluteLayout .Tamaño automático));  
  
        AbsoluteLayout .SetLayoutBounds (label2,  
            nuevo Rectángulo (0.5, 1 - t, AbsoluteLayout .Tamaño automático, AbsoluteLayout .Tamaño automático));  
  
        return true ;  
    }  
}
```

Los OnTimerTick controlador calcula un tiempo transcurrido desde que se inició el programa y lo convierte en un valor que t (por el tiempo) que va de 0 a 1 cada dos segundos. El segundo cálculo de t hace que aumente de 0 a 1 y luego disminuye de nuevo a 0 cada dos segundos. Este valor se pasa directamente a la Rectángulo constructor en los dos AbsoluteLayout.SetLayoutBounds llamadas. El resultado es que la primera Etiqueta mueve horizontalmente a través del centro de la pantalla y parece que rebota en los lados derecho e izquierdo. El segundo Etiqueta mueve verticalmente hacia arriba y hacia abajo el centro de la pantalla y parece rebotar en la parte superior e inferior:



Los dos Etiqueta vistas reúnen brevemente en el centro de cada segundo, como la captura de pantalla de Windows 10 móvil confirma.

De aquí en adelante, las páginas de nuestras aplicaciones Xamarin.Forms llegarán a ser más activo y animado y dinámico. En el siguiente capítulo, verá cómo las vistas interactivas de Xamarin.Forms establecer un medio de comunicación entre el usuario y la aplicación.

capítulo 15

La interfaz interactiva

La interactividad es la característica definitoria de la informática moderna. Las muchas vistas interactivas que Xamarin.Forms instrumentos responden a gestos táctiles como tocar y arrastrar, y algunos incluso leer las pulsaciones del teclado desde el teclado virtual del teléfono.

Estas vistas interactivas incorporan paradigmas que son familiares para los usuarios, e incluso tienen nombres que son familiares para los programadores: los usuarios pueden ejecutar comandos con Botón, especificar un número de un rango de valores con deslizador y paso a paso, introducir texto desde el teclado del teléfono utilizando Entrada y Editor, y seleccionar elementos de una colección con Picker, ListView, y TableView.

Este capítulo está dedicado a demostrar que muchas de estas vistas interactivas.

Ver descripción general

Xamarin.Forms define 20 clases instanciables que se derivan de Ver pero no de Diseño. Que ya ha visto seis de estas clases en los capítulos anteriores: Etiqueta, BoxView, Botón, Imagen, ActivityIndicator, y Barra de progreso.

Este capítulo se centra en ocho puntos de vista que permiten al usuario seleccionar o interactuar con tipos de datos .NET básicos:

Tipo de datos	Puntos de vista
Doble	Deslizador, paso a paso
Boole	Cambiar
Cuerda	Entrada, Editor, SearchBar
Fecha y hora	DatePicker, TimePicker

Estos puntos de vista son a menudo las representaciones visuales de los elementos de datos subyacentes. En el siguiente capítulo, usted comenzará a explorar el enlace de datos, que es una característica de Xamarin.Forms que une las propiedades de puntos de vista con propiedades de otras clases para que estos puntos de vista y los datos subyacentes se puedan estructurar de correspondencias.

Cuatro de las seis vistas restantes se tratan en capítulos posteriores. En el capítulo 16, "El enlace de datos", verá lo siguiente:

- WebView, para visualizar páginas web o HTML.

Capítulo 19, "vistas Collection" abarca estos tres puntos de vista:

- Recogedor, cuerdas seleccionables para las opciones del programa.
- Vista de la lista, una lista desplegable de elementos de datos del mismo tipo.

- TableView, una lista de elementos separados en categorías, que es lo suficientemente flexible para ser utilizado para los datos, formularios, menús o ajustes.

Dos puntos de vista no están cubiertos en esta edición de este libro:

- Mapa, una pantalla de mapa interactivo.
- OpenGLView, que permite a un programa para visualizar 2-D y 3-D gráficos mediante el uso de la biblioteca de gráficos abierto.

Deslizador y paso a paso

Ambos deslizador y paso a paso permitir al usuario seleccionar un valor numérico de un rango. Tienen interfaces de programación casi idénticos, pero incorporan muy diferentes paradigmas visuales e interactivos.

fundamentos deslizante

los Xamarin.Forms deslizador es una barra horizontal que representa un rango de valores entre un mínimo a la izquierda y un máximo a la derecha. (Los Xamarin.Forms deslizador no soporta una orientación vertical.) El usuario selecciona un valor en el deslizador un poco diferente en las tres plataformas: En los dispositivos iOS, el usuario arrastra una ronda “pulg” a lo largo de la barra horizontal. El móvil Android y Windows 10

deslizador También tienen vistas pulgares, pero son demasiado pequeños para un objetivo tacto, y el usuario puede simplemente toque en la barra horizontal, o arrastrar un dedo a una ubicación específica.

los deslizador define tres propiedades públicas de tipo doble, llamado Mínimo máximo, y Valor.

Cada vez que el Valor cambios de propiedad, la deslizador dispara una ValueChanged evento que indica el nuevo valor.

Cuando se muestra una deslizador usted querrá un poco de relleno en la parte izquierda y derecha para evitar la deslizador desde que se extiende hasta los bordes de la pantalla. El archivo XAML en el SliderDemo se aplica el programa de

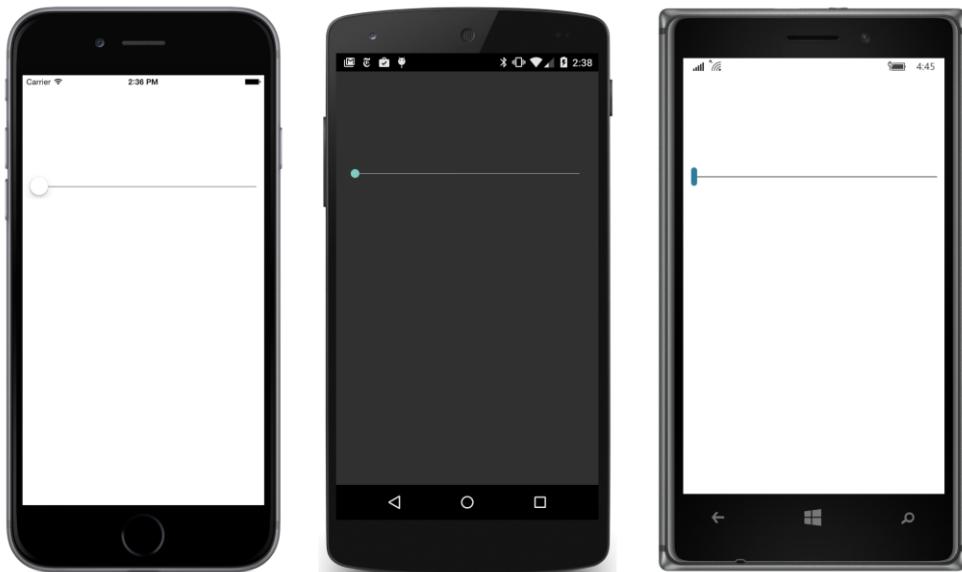
Relleno al StackLayout, que es el padre a la vez una deslizador y una Etiqueta que está destinada a mostrar el valor actual de la deslizador:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " SliderDemo.SliderDemoPage " >

    < StackLayout Relleno = " 10, 0 " >
        < deslizador VerticalOptions = " CenterAndExpand "
            ValueChanged = " OnSliderValueChanged " />

        < Etiqueta x: Nombre = " etiqueta "
            Tamaño de fuente = " Grande "
            HorizontalOptions = " Centrar "
            VerticalOptions = " CenterAndExpand " />
    </ StackLayout >
</ Página de contenido >
```

Cuando el programa se pone en marcha, el Etiqueta muestra nada, y el deslizador pulgar se coloca en el extremo izquierdo:



No ajustar HorizontalOptions sobre el deslizador a Inicio, Centro, o Fin sin establecer también WidthRequest a un valor explícito, o la deslizador colapsará en una anchura muy pequeña o incluso inutilizable.

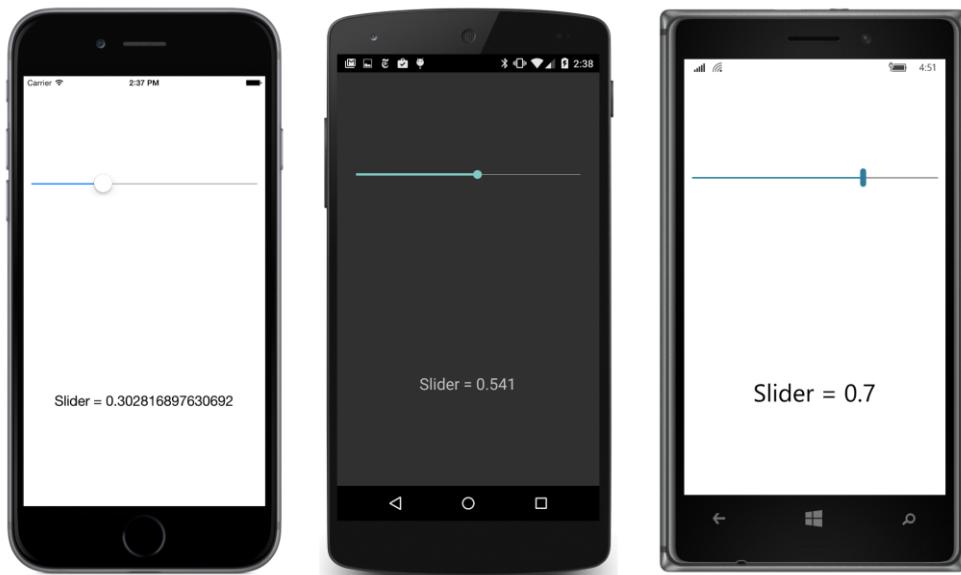
los deslizador notifica el código de cambios en el Valor propiedad por el disparo de la ValueChanged evento. El evento se dispara si Valor se cambia mediante programación o por la manipulación del usuario. Aquí está la SliderDemo archivo de código subyacente con el controlador de eventos:

```
pública clase parcial SliderDemoPage : Pagina de contenido
{
    pública SliderDemoPage ()
    {
        InitializeComponent ();
    }

    vacío OnSliderValueChanged ( objeto remitente, ValueChangedEventArgs args )
    {
        label.text = Cuerda .Formato( "Slider = {0}" , Args.NewValue );
    }
}
```

Como de costumbre, el primer argumento al controlador de eventos es el objeto de disparar el evento, en este caso el deslizador, y el segundo argumento proporciona más información acerca de este evento. El controlador para ValueChanged es de tipo Manejador de sucesos <ValueChangedEventArgs>, lo que significa que el segundo argumento al controlador es una ValueChangedEventArgs objeto. ValueChangedEventArgs define dos propiedades

de tipo doble llamado **Valor antiguo** y **Nuevo valor**. Este controlador particular utiliza simplemente **Nuevo valor** en una cadena que se fija a la **Texto** propiedad de la Etiqueta:



Un poco de experimentación revela que el valor por defecto Mínimo y Máximo ajustes para deslizador son 0

1. En el momento de este capítulo se está escribiendo, la deslizador en las plataformas de Windows tiene un incremento predeterminado de 0,1. Para otros ajustes de Mínimo y Máximo, el deslizador está restringido a incrementos de 10 o pasos de 1, lo que sea menor. (A más flexible deslizador se presenta en el capítulo 27, "procesadores personalizados.")

Si no está satisfecho con el excesivo número de decimales que aparecen en la pantalla de iOS, se puede reducir el número de decimales con una especificación de formato de `String.Format`:

```
vacío OnSliderValueChanged ( objeto remitente, ValueChangedEventArgs args)
{
    label.text = Cuerda .Formato( "Slider = {0: F2}" , Args.NewValue);
}
```

Esta no es la única manera de escribir la `ValueChanged` entrenador de animales. Una implementación alternativa consiste en colar el primer argumento a una deslizador objeto y luego el acceso a la `Valor` propiedad directa:

```
vacío OnSliderValueChanged ( objeto remitente, ValueChangedEventArgs args)
{
    deslizador deslizador = ( deslizador )remitente;
    label.text = Cuerda .Formato( "Slider = {0}" , Slider.Value);
}
```

Utilizando el `remitente` argumento es un buen método si va a compartir el controlador de eventos entre los múltiples deslizador puntos de vista. En el momento en el `ValueChanged` controlador de eventos se llama, el `Valor` propiedad ya tiene su nuevo valor.

Puede establecer la Mínimo y Máximo propiedades de la deslizador a cualquier valor negativo o positivo, con la condición de que Máximo es siempre mayor que Mínimo. Por ejemplo, intente lo siguiente:

```
<deslizador ValueChanged = "OnSliderValueChanged"
    Máximo = "100"
    VerticalOptions = "CenterAndExpand" />
```

Ahora el deslizador rango de los valores de 0 a 100.

Errores comunes

Supongamos que se desea que el deslizador valor en un rango de 1 a 100. Se puede establecer tanto Mínimo y Máximo

Me gusta esto:

```
<deslizador ValueChanged = "OnSliderValueChanged"
    Mínimo = "1"
    Máximo = "100"
    VerticalOptions = "CenterAndExpand" />
```

Sin embargo, cuando se ejecuta la nueva versión del programa, una ArgumentException se eleva con la explicación de texto "Valor fue un valor no válido para el mínimo." ¿Qué significa eso?

Cuando el analizador XAML se encuentra con el deslizador etiqueta, una deslizador se crea una instancia, y luego las propiedades y los eventos se establecen en el orden en el que aparecen en la deslizador etiqueta. Pero cuando el Mínimo propiedad se establece en 1, el Máximo Ahora es igual al valor Mínimo valor. Eso no puede ser. Los Máximo propiedad debe ser *mayor* que la Mínimo. Los deslizador señales de este problema elevando una excepción.

Interna a la deslizador clase, el Mínimo y Máximo valores se comparan en un método de devolución de llamada se establece en el validateValue argumento de la BindableProperty.Create llamadas a métodos que crean la

Mínimo y Máximo propiedades enlazables. Los validateValue retornos de devolución de llamada cierto Si Mínimo es menor que Máximo, lo que indica que los valores son válidos. Un valor de retorno de falso de esta devolución de llamada desencadena la excepción. Esta es la forma estándar que implementan propiedades enlazables comprobaciones de validez.

Esto no es un problema específico de XAML. También sucede si usted instancia e inicializar el deslizador propiedades en este orden en el código. La solución es invertir el orden que Mínimo y Máximo se establecen. En primer lugar establecer el Máximo propiedad a 100. Es legal porque ahora el rango es entre 0 y 100. A continuación, establezca la Mínimo propiedad a 1:

```
<deslizador ValueChanged = "OnSliderValueChanged"
    Máximo = "100"
    Mínimo = "1"
    VerticalOptions = "CenterAndExpand" />
```

Sin embargo, esto se traduce en otro error de tiempo de ejecución. Ahora se trata de una Excepcion de referencia nula en el Valor- cambiado entrenador de animales. ¿Porqué es eso?

Los Valor propiedad de la deslizador debe estar dentro de la gama de Mínimo y Máximo valores, así que cuando la Mínimo propiedad se establece en 1, el deslizador ajustar automáticamente su Valor propiedad a 1.

Internamente, Valor se ajusta en un método de devolución de llamada se establece en el coerceValue argumento de la BindableProperty.Create llamadas a métodos para la Mínimo máximo, y Valor propiedades. El método de devolución de llamada devuelve un valor ajustado de la propiedad que está siendo ajustado después de ser sometido a esta coacción. En este ejemplo, cuando Mínimo se pone a 1, la coerceValue método establece el control deslizante de Valor propiedad a 1, y la coerceValue devolución de llamada devuelve el nuevo valor de Mínimo, que permanece en el valor 1.

Sin embargo, como resultado de la coacción, la Valor propiedad ha cambiado, y esto hace que el Valor cambiado evento al fuego. Los ValueChanged manejador en el archivo de código subyacente intenta establecer el Texto propiedad de la Etiqueta, pero el analizador XAML aún no ha instanciado el Etiqueta elemento. Los etiqueta campo es nulo.

Hay un par de soluciones a este problema. La solución más segura y más general es para comprobar si hay una nulo valor por etiqueta justo en el controlador de eventos:

```
vacio OnSliderValueChanged ( objeto remitente, ValueChangedEventArgs args)
{
    Si (Etiqueta! = nulo)
    {
        label.text = Cuerda .Formato( "Slider = {0}" , Args.NewValue);
    }
}
```

Sin embargo, también se puede solucionar el problema moviendo la asignación de la ValueChanged evento en la etiqueta para después de la Máximo y Mínimo propiedades se han establecido:

```
<deslizador Máximo = "100"
    Mínimo = "1"
    ValueChanged = "OnSliderValueChanged"
    VerticalOptions = "CenterAndExpand" />
```

Los Valor propiedad todavía es forzado a 1 después de la Mínimo propiedad está establecida, pero la ValueChanged controlador de eventos aún no se ha asignado, por lo que no hay ningún evento disparado.

Vamos a suponer que la deslizador tiene el rango predeterminado de 0 a 1. Es posible que deseé el Etiqueta para mostrar el valor inicial de la deslizador cuando el programa se inicia por primera vez. Se podría inicializar el Texto propiedad de la Etiqueta a "Slider = 0" en el archivo XAML, pero si alguna vez quiere cambiar el texto a algo un poco diferente, que habrá necesidad de cambiarlo en dos lugares.

Usted puede tratar de darle la deslizador un nombre de deslizador en el archivo XAML y luego añadir algo de código para el constructor:

```
público SliderDemoPage ()
{
    InitializeComponent ();

    slider.Value = 0;
}
```

Todos los elementos en el archivo XAML se han creado e inicializado cuando InitializeComponent devuelve, por lo que si este código hace que el deslizador para disparar una ValueChanged acontecimiento, que no debería ser un problema.

Pero no va a funcionar. El valor de la deslizador ya es 0, por lo que establecer a 0 de nuevo no hace nada. Usted podría intentar esto:

```
p\xedblico SliderDemoPage ()
{
    InitializeComponent ();

    slider.Value = 1;
    slider.Value = 0;
}
```

Que funcionará. Sin embargo, es posible que desee añadir un comentario en el código para que otro programador no quita después la instrucción que establece Valor a 1, ya que parece ser innecesaria.

O bien, podría simular un evento llamando directamente al controlador. Los dos argumentos a la ValueChangedEventArgs constructor son el valor antiguo y el nuevo valor (en ese orden), pero el EventHandlerSliderValueChanged manejador utiliza sólo el Nuevo valor propiedad, por lo que no importa lo que el otro argumento es o si son iguales:

```
p\xedblico clase parcial SliderDemoPage : Pagina de contenido
{
    p\xedblico SliderDemoPage ()
    {
        InitializeComponent ();

        OnSliderValueChanged ( nulo , nuevo ValueChangedEventArgs (0, 0));
    }

    vac\xedo OnSliderValueChanged ( objeto remitente, ValueChangedEventArgs args)
    {
        label.text = Cuerda .Formato( "Slider = {0}" , Args.NewValue);
    }
}
```

Eso funciona así. Pero recuerde que debe establecer los argumentos para la llamada a OnSliderValueChanged por lo que están de acuerdo con lo que el manejador de espera. Si se ha sustituido el cuerpo manejador con el código que arroja el remitente argumento de la deslizador objeto, este caso es necesario un primer argumento válido en el EventHandlerSliderValueChanged llamada.

Los problemas relacionados con el controlador de eventos desaparecen cuando se conecta el Etiqueta con el deslizador mediante el uso de enlaces de datos, que con lo siguiente: en el capítulo siguiente. No obstante, deberá configurar las propiedades de la deslizador en el orden correcto, pero usted experimentará ninguno de los problemas con el controlador de eventos debido a que el controlador de eventos se ha ido.

selección de color deslizador

He aquí un programa llamado RgbSliders que contiene tres deslizador elementos para la selección de componentes rojo, verde y azul de una Color. Un estilo implícito para deslizador establece el Máximo valor a 255:

< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "

```

    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " RgbSliders.RgbSlidersPage " >
< ContentPage.Padding >
    < OnPlatform x: TypeArguments = " Espesor " >
        iOS = " 10, 20, 10, 10 "
        Androide = " 10, 0, 10, 10 "
        WinPhone = " 10, 0, 10, 10 " />
</ ContentPage.Padding >

< StackLayout >
    < StackLayout.Resources >
        < ResourceDictionary >
            < Estilo Tipo de objetivo = " deslizador " >
                < Setter Propiedad = " Máximo " Valor = " 255 " />
            </ Estilo >

            < Estilo Tipo de objetivo = " Etiqueta " >
                < Setter Propiedad = " Tamaño de fuente " Valor = " Grande " />
                < Setter Propiedad = " HorizontalTextAlignment " Valor = " Centrar " />
            </ Estilo >
        </ ResourceDictionary >
    </ StackLayout.Resources >

    < deslizador x: Nombre = " redSlider "
        ValueChanged = " OnSliderValueChanged " />

    < Etiqueta x: Nombre = " etiqueta roja " />

    < deslizador x: Nombre = " greenSlider "
        ValueChanged = " OnSliderValueChanged " />

    < Etiqueta x: Nombre = " etiqueta verde " />

    < deslizador x: Nombre = " blueSlider "
        ValueChanged = " OnSliderValueChanged " />

    < Etiqueta x: Nombre = " etiqueta azul " />

    < BoxView x: Nombre = " boxView "
        VerticalOptions = " FillAndExpand " />
</ StackLayout >
</ Pagina de contenido >

```

los deslizador elementos alternativos con tres Etiqueta elementos para mostrar sus valores, y el StackLayout fuera concluye con una BoxView para mostrar el color resultante.

El constructor del archivo de código subyacente inicializa el deslizador ajustes a 128 para un gris medio. la compartido ValueChanged controlador comprueba para ver qué deslizador ha cambiado, y por lo tanto que Etiqueta necesita ser actualizado, y luego calcula un nuevo color para el BoxView:

```

pública clase parcial RgbSlidersPage : Pagina de contenido
{
    pública RgbSlidersPage ()

```

```
{  
    InitializeComponent();  
  
    redSlider.Value = 128;  
    greenSlider.Value = 128;  
    blueSlider.Value = 128;  
}  
  
vacío OnSliderValueChanged ( objeto remitente, ValueChangedEventArgs args)  
{  
    Si (Remitente == redSlider)  
    {  
        redLabel.Text = Cuerda .Formato( "Red = {0: X2}" , ( Ent ) RedSlider.Value);  
    }  
    else if (Remitente == greenSlider)  
    {  
        greenLabel.Text = Cuerda .Formato( "Green = {0: X2}" , ( Ent ) GreenSlider.Value);  
    }  
    else if (Remitente == blueSlider)  
    {  
        blueLabel.Text = Cuerda .Formato( "Azul = {0: X2}" , ( Ent ) BlueSlider.Value);  
    }  
  
    boxView.Color = Color .FromRgb (( Ent ) RedSlider.Value,  
                                    ( Ent ) GreenSlider.Value,  
                                    ( Ent ) BlueSlider.Value);  
}  
}
```

Estrictamente hablando, el Si y más declaraciones continuación no son necesarios. El código puede simplemente establecer las tres etiquetas independientemente de la corredera está cambiando. El controlador de eventos accede a los tres deslizadores de todos modos para el establecimiento de un nuevo color:



Puede activar el teléfono de lado, pero el BoxView se hace mucho más corto, sobre todo en el dispositivo móvil de Windows 10, donde el deslizador parece tener una altura vertical más allá de lo que se requiere. Una vez el Cuadrícula se introduce en el capítulo 18, verá cómo se hace más fácil para las aplicaciones respondan a los cambios de orientación.

El seguimiento **TextFade** programa utiliza un único deslizador para controlar el Opacidad y la posición horizontal de dos Etiqueta elementos de una AbsoluteLayout. En el diseño inicial, tanto Etiqueta Los elementos se colocan en la parte central izquierda de la AbsoluteLayout, pero el segundo tiene su Opacidad se pone a 0:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " TextFade.TextFadePage "
    Relleno = " 10, 0, 10, 20 " >

    < StackLayout >
        < AbsoluteLayout VerticalOptions = " CenterAndExpand " >
            < Etiqueta x: Nombre = " label1 " 
                Texto = " TEXTO "
                Tamaño de fuente = " Grande "
                AbsoluteLayout.LayoutBounds = " 0, 0, 0,5 "
                AbsoluteLayout.LayoutFlags = " PositionProportional " />

            < Etiqueta x: Nombre = " label2 " 
                Texto = " DESCOLORARSE "
                Tamaño de fuente = " Grande "
                Opacidad = " 0 "
                AbsoluteLayout.LayoutBounds = " 0, 0, 0,5 "
                AbsoluteLayout.LayoutFlags = " PositionProportional " />
        </ AbsoluteLayout >
    </ StackLayout >
```

```
</deslizador ValueChanged = "OnSliderValueChanged" />

</StackLayout>
</Página de contenido>
```

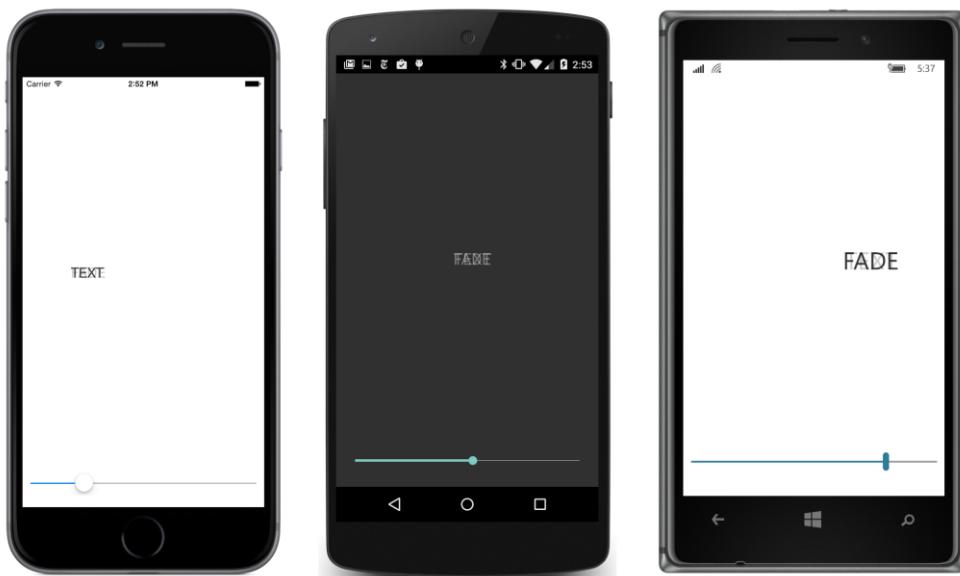
los deslizador controlador de eventos se mueve tanto Etiqueta elementos de izquierda a derecha en la pantalla. El posicionamiento proporcional ayuda mucho aquí porque el deslizador valores van de 0 a 1, lo que resulta en la Etiqueta elementos estando colocados progresivamente desde el extremo izquierdo al extremo derecho de la pantalla:

```
pública clase parcial TextFadePage : Página de contenido
{
    pública TextFadePage ()
    {
        InitializeComponent ();
    }

    vacío OnSliderValueChanged ( objeto remitente, ValueChangedEventArgs args)
    {
        AbsoluteLayout .SetLayoutBounds (label1,
            nuevo Rectángulo (Args.NewValue, 0,5, AbsoluteLayout .Tamaño automático,
                AbsoluteLayout .Tamaño automático));
        AbsoluteLayout .SetLayoutBounds (label2,
            nuevo Rectángulo (Args.NewValue, 0,5, AbsoluteLayout .Tamaño automático,
                AbsoluteLayout .Tamaño automático));

        label1.Opacity = 1 - args.NewValue;
        label2.Opacity = args.NewValue;
    }
}
```

Al mismo tiempo, la Opacidad valores se establecen de modo que uno Etiqueta parece desvanecerse en el otro como dos etiquetas se mueven por la pantalla:



La diferencia de pasos

los paso a punto de vista tiene casi la misma interfaz de programación como el deslizador: Tiene Mínimo, Máximo, y Valor inmuebles del tipo doble y dispara un ValueChanged controlador de eventos.

sin embargo, el Máximo propiedad de paso a punto tiene un valor predeterminado de 100, y paso a punto También añade una Incremento propiedad con un valor predeterminado de 1. La paso a punto visuales consisten únicamente de dos botones marcados con signos menos y más. Prensas de esos dos botones cambian el valor incremental entre Mínimo a Máximo basado en el Incremento propiedad.

A pesar de que Valor y otras propiedades de paso a punto son de tipo doble, paso a punto a menudo se utiliza para la selección de valores enteros. Es probable que no desea que el valor de $((\text{Máximo} - \text{Mínimo}) / \text{Incremento})$ sea tan alta como 100, como sugieren los valores por defecto. Si se mantiene pulsado el dedo sobre uno de los botones, se activará la repetición typematic en iOS, pero no en Android o Windows Mobile 10. A menos que su programa proporciona otra forma para que el usuario cambie la paso a punto valor (tal vez con un texto Entrada Vista), que no quiere forzar a que el usuario pulse un botón 100 veces para obtener de Mínimo a Máximo.

los StepperDemo programa establece el Máximo propiedad de la paso a punto a 10 y utiliza el Paso- por como una ayuda para el diseño rudimentario en la determinación de un ancho de borde óptimo para una Botón frontera. los Botón en la parte superior de la StackLayout es únicamente para fines de visualización y tiene la configuración de propiedades necesarias de Color de fondo y Color del borde para permitir la visualización de la frontera en Android y Windows Mobile 10.

los paso a punto es el último hijo de la siguiente StackLayout. Entre los Botón y paso a punto son un par de Etiqueta elementos para la visualización de la corriente paso a punto valor:

```

< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " StepperDemo.StepperDemoPage " >

    < StackLayout >
        < Botón x: Nombre = " botón "
            Texto = " Botón de muestra "
            Tamaño de fuente = " Grande "
            HorizontalOptions = " Centrar "
            VerticalOptions = " CenterAndExpand " >
            < Button.BackgroundColor >
                < OnPlatform x: TypeArguments = " Color "
                    Androide = "# 404040 " />
            </ Button.BackgroundColor >
            < Button.BorderColor >
                < OnPlatform x: TypeArguments = " Color "
                    Androide = "# C0C0C0 "
                    WinPhone = " Negro " />
            </ Button.BorderColor >
        </ Botón >
    < StackLayout VerticalOptions = " CenterAndExpand " >

        < StackLayout Orientación = " Horizontal "
            HorizontalOptions = " Centrar " >
            < StackLayout.Resources >
                < ResourceDictionary >
                    < Estilo Tipo de objetivo = " Etiqueta " >
                        < Setter Propiedad = " Tamaño de fuente " Valor = " Medio " />
                    </ Estilo >
                </ ResourceDictionary >
            </ StackLayout.Resources >
            < Etiqueta Texto = " Botón Ancho del borde = " />
            < Etiqueta x: Nombre = " etiqueta " />
        </ StackLayout >

        < paso a paso x: Nombre = " paso a paso "
            Máximo = " 10 "
            ValueChanged = " OnStepperValueChanged "
            HorizontalOptions = " Centrar " />
    </ StackLayout >
</ StackLayout >
</ Pagina de contenido >

```

los Etiqueta mostrando el paso a paso valor se inicializa desde el constructor del archivo de código subyacente. Con cada cambio en el Valor propiedad de la paso a paso, el controlador de eventos muestra el nuevo valor y establece el Botón ancho del borde:

```

pública clase parcial StepperDemoPage : Pagina de contenido
{
    pública StepperDemoPage ()
    {

```

```
InitializeComponent();  
  
    // Inicializar pantalla.  
    OnStepperValueChanged (paso a paso, nulo);  
}  
  
void OnStepperValueChanged ( objeto remitente, ValueChangedEventArgs args)  
{  
    paso a paso stepper = (paso a paso)remitente;  
    button.BorderWidth = stepper.Value;  
    label.text = stepper.Value.ToString ("F0");  
}  
}
```



Switch y CheckBox

Los programas de aplicación a menudo necesitan entrada booleana del usuario, lo que requiere un poco de camino para el usuario cambiar una opción de programa en Sí o No, Sí o No, Verdadero o Falso, o como le quieran pensar en él. En Xamarin.Forms, se trata de una vista llamada la Cambiar.

conceptos básicos de conmutación

Cambiar define una sola propiedad por su cuenta, el nombre IsToggled de tipo bool, y se dispara el Ataviar-
GLED evento para indicar un cambio en esta propiedad. En el código, que podría estar inclinado a dar una Cambiar un nombre de cambiar, pero eso
es una palabra clave # C, por lo que desea elegir otra cosa. En XAML, sin embargo, se puede establecer el x: Nombre atribuir a cambiar, y el
análizador XAML inteligentemente crear un campo denominado

@cambiar, que es como C# permite definir un nombre de variable utilizando una palabra clave de C#.

los **SwitchDemo** programa crea dos Cambiar elementos con dos etiquetas de identificación: "Cursiva" y "negrita". Cada Cambiar tiene su propio controlador de eventos, lo que da formato a la mayor Etiqueta en la parte inferior de la StackLayout:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns:x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " SwitchDemo.SwitchDemoPage " >

    < StackLayout Relleno = " 10, 0 " >
        < StackLayout HorizontalOptions = " Centrar "
            VerticalOptions = " CenterAndExpand " >
            < StackLayout Orientación = " Horizontal "
                HorizontalOptions = " Fin " >
                    < Etiqueta Texto = " Ítálico: "
                        VerticalOptions = " Centrar " />
                    < Cambiar toggled = " OnItalicSwitchToggled "
                        VerticalOptions = " Centrar " />
                </ StackLayout >
        </ StackLayout Orientación = " Horizontal "
            HorizontalOptions = " Fin " >
            < Etiqueta Texto = " negrita: "
                VerticalOptions = " Centrar " />
            < Cambiar toggled = " OnBoldSwitchToggled "
                VerticalOptions = " Centrar " />
        </ StackLayout >
    </ StackLayout >

    < Etiqueta x: Nombre = " etiqueta "
        Texto =
    " Sólo un pequeño paso de texto de ejemplo que se pueden formatear
    en cursiva o negrita alternando los dos elementos de comutación. "
        Tamaño de fuente = " Grande "
        HorizontalTextAlignment = " Centrar "
        VerticalOptions = " CenterAndExpand " />

    </ StackLayout >
</ Página de contenido >
```

los toggled controlador de eventos tiene un segundo argumento de ToggledEventArgs, que tiene una Valor propiedad de tipo bool que indica el nuevo estado de la IsToggled propiedad. Los controladores de eventos en **SwitchDemo** utilizar este valor para establecer o borrar la particular FontAttributes bandera en el FontAttributes propiedad de la larga Etiqueta:

```
público clase parcial SwitchDemoPage : Página de contenido
{
    público SwitchDemoPage ()
    {
        InitializeComponent ();
    }
}
```

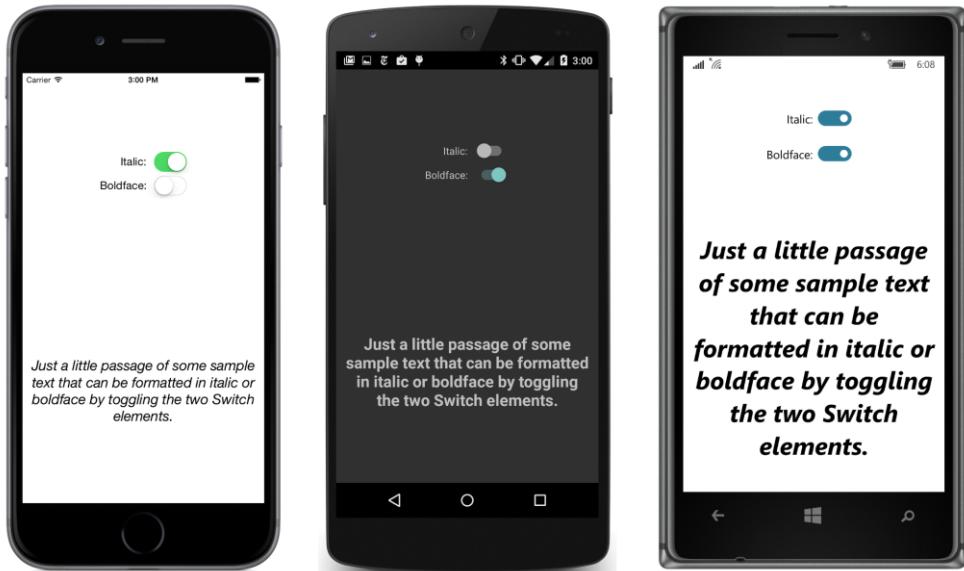
```

    void OnItalicSwitchToggled ( objeto remitente, ToggledEventArgs args)
    {
        Si (Args.Value)
        {
            label.FontAttributes |= FontAttributes .Itálico;
        }
        más
        {
            label.FontAttributes &= ~ FontAttributes .Itálico;
        }
    }

    void OnBoldSwitchToggled ( objeto remitente, ToggledEventArgs args)
    {
        Si (Args.Value)
        {
            label.FontAttributes |= FontAttributes .Negrita;
        }
        más
        {
            label.FontAttributes &= ~ FontAttributes .Negrita;
        }
    }
}

```

los Cambiar tiene un aspecto diferente en las tres plataformas:



Observe que el programa se alinea las dos Cambiar puntos de vista, lo que le da un aspecto más atractivo, pero que también significa que las etiquetas de texto son necesariamente algo desalineada. Para lograr este formato, el archivo XAML pone cada uno del par de Etiqueta y Cambiar elementos en una horizontal StackLayout.

cada horizontal StackLayout tiene su HorizontalOptions ajustado a Fin, que alinea cada StackLayout a la derecha, y un padre StackLayout centra la colección de etiquetas y los commutadores de la pantalla con una HorizontalOptions ajuste de Centrar. Dentro de la horizontal StackLayout, Ambos puntos de vista tienen su VerticalOptions propiedades ajustado a Centrar. Si el Cambiar es más alto que el Etiqueta, entonces el Label se centra verticalmente con respecto a la Cambiar. Pero si el Etiqueta es más alto que el Cambiar, el Cambiar también se centra verticalmente con respecto a la Etiqueta.

Un CheckBox tradicional

En entornos gráficos más tradicionales, el objeto de interfaz de usuario que permite a los usuarios elegir un valor booleano se llama una Caja, por lo general con un texto que contiene una caja que puede estar vacío o lleno de una X o una marca de verificación. Una de las ventajas de la Caja sobre el Cambiar es que el identificador de texto es parte de la visual y no necesita ser añadido con un separado Etiqueta.

Una forma de crear vistas personalizadas en Xamarin.Forms es escribiendo clases especiales llamadas *extracción de grasas* que son específicos de cada plataforma y que los puntos de vista de referencia en cada plataforma. Eso se demuestra en el capítulo 27.

Sin embargo, también es posible crear vistas personalizadas derecha en Xamarin.Forms ensamblando una vista desde otros puntos de vista. En primer lugar, derive una clase de ContentView, establecer su Contenido propiedad a un StackLayout (Por ejemplo) y, a continuación, añadir uno o más puntos de vista sobre eso. (Usted vio un ejemplo de esta técnica en el ColorView clase en el capítulo 8.) Es posible que también necesita definir una o más propiedades, y posiblemente algunos eventos, pero usted quiere tomar ventaja de la infraestructura establecida por la enlazable bindableObject y BindableProperty clases. Eso permite que sus propiedades a ser de estilo y siendo blanco de enlaces de datos.

UN Caja consta de sólo dos Etiqueta elementos en una ContentView: uno Etiqueta muestra el texto asociado con el Caja, mientras que la otra muestra un cuadro. UN TapGestureRecognizer detecta cuando el Caja se toca.

UN Caja clase ya ha sido añadido a la **Xamarin.FormsBook.Toolkit** biblioteca que se incluye en el código descargable para este libro. He aquí cómo usted lo haría por su cuenta:

En Visual Studio, puede seleccionar Página forma Xaml desde el Agregar ítem nuevo caja de diálogo. Sin embargo, esto crea una clase que deriva de Pagina de contenido cuando usted realmente desea una clase que deriva de EstatentView. Simplemente cambie el elemento raíz del archivo XAML de Pagina de contenido a ContentView, y cambiar la clase base en el archivo de código subyacente de Pagina de contenido a ContentView.

En Xamarin de estudio, sin embargo, sólo tiene que elegir ContentView forma Xaml desde el Archivo nuevo diálogo.

Aquí está el archivo CheckBox.xaml:

```
<ContentPage xmlns = " http://xamarin.com/schemas/2014/forms "
             xmlns:x = " http://schemas.microsoft.com/winfx/2009/xaml "
             x:Class = " Xamarin.FormsBook.Toolkit.CheckBox " >
```

```

< StackLayout Orientación = "Horizontal" >
    < Etiqueta x: Nombre = "box_label" Texto = "\u2610;" />
    < Etiqueta x: Nombre = "text_label" />
</ StackLayout >

< ContentView.GestureRecognizers >
    < TapGestureRecognizer apropiado = "OnCheckBoxTapped" />
</ ContentView.GestureRecognizers >
</ ContentView >

```

Ese carácter Unicode \ u2610 se llama el personaje Urna de voto, y es sólo una casilla vacía. Carácter \ u2611 es una urna con el cheque, mientras que \ u2612 es una urna con X. Para indicar un estado marcada, Caja archivo de código subyacente establece el Texto propiedad de boxLabel a \ u2611 (como se verá en breve).

El archivo de código subyacente de Caja define tres propiedades:

- Texto
- Tamaño de fuente
- Está chequeado

Caja también define un evento denominado IsCheckedChanged.

Debería Caja también definir FontAttributes y Familia tipográfica propiedades como Etiqueta y Peritoneada ¿hacer? Tal vez, pero estas propiedades adicionales no son tan cruciales para las vistas dedicadas a la interacción del usuario.

Los tres de las propiedades que Caja define están respaldados por propiedades enlazables. El archivo de código subyacente crea los tres BindableProperty objetos, y los manipuladores de propiedad-cambiado se definen como funciones lambda dentro de estos métodos.

Tenga en cuenta que los manipuladores de propiedad-cambiado son estáticos, por lo que necesitan para lanzar el primer argumento a una Caja objeto hacer referencia a las propiedades de la instancia y eventos en la clase. El manejador de PropertyChanged Está chequeado es responsable de cambiar el carácter que representa el estado marcado y sin marcar y disparar el IsCheckedChanged evento:

```

espacio de nombres Xamarin.FormsBook.Toolkit
{
    público clase parcial Caja : ContentView
    {
        sólo lectura estática pública BindableProperty TextProperty =
        BindableProperty .Crear(
            "Texto",
            tipo de ( cuerda ),
            tipo de ( Caja ),
            nulo ,
            PropertyChanged: (enlazable, oldValue, newValue) =>
            {
                (( Caja ) Enlazable) .textLabel.Text = ( cuerda )nuevo valor;
            });
    }
}

```

```
sólo lectura estática pública BindablePropertyFontSizeProperty =  
BindableProperty.Crear(  
    "Tamaño de fuente",  
    tipo de ( doble ),  
    tipo de ( Caja ),  
    Dispositivo.GetNamedSize ( NamedSize.Defecto, tipo de ( Etiqueta )),  
    PropertyChanged: (enlazable, oldValue, newValue) =>  
    {  
        Caja casilla de verificación = ( Caja ) Enlazable;  
        checkbox.boxLabel.FontSize = ( doble )nuevo valor;  
        checkbox.textLabel.FontSize = ( doble )nuevo valor;  
    });  
  
sólo lectura estática pública BindablePropertyIsCheckedProperty =  
BindableProperty.Crear(  
    "Está chequeado",  
    tipo de ( bool ),  
    tipo de ( Caja ),  
    falso ,  
    PropertyChanged: (enlazable, oldValue, newValue) =>  
    {  
        // Establecer el gráfico.  
        Caja casilla de verificación = ( Caja ) Enlazable;  
        checkbox.boxLabel.Text = ( bool )nuevo valor ? "\u2611" : "\u2610";  
  
        // desencadenar el evento.  
        Controlador de eventos < bool > EventHandler = checkbox.CheckedChanged;  
        Si (EventHandler!=nulo)  
        {  
            eventHandler (casilla de verificación, ( bool )nuevo valor);  
        }  
    });  
  
evento público Controlador de eventos < bool > CheckedChanged;  
  
público Caja()  
{  
    InitializeComponent ();  
}  
  
public string Texto  
{  
    conjunto {EstablecerValor (TextProperty, valor );}  
    obtener {regreso ( cuerda ) GetValue (TextProperty);}  
}  
  
[ TypeConverter ( tipo de ( FontSizeConverter ))]  
pública doble Tamaño de fuente  
{  
    conjunto {EstablecerValor (FontSizeProperty, valor );}  
    obtener {regreso ( doble ) GetValue (FontSizeProperty);}  
}
```

```

public bool Está chequeado
{
    conjunto {EstablecerValor (IsCheckedProperty, valor);}

    obtener {regreso (bool) GetValue (IsCheckedProperty);}

}

// manipulador TapGestureRecognizer.
vacío OnCheckBoxTapped ( objeto remitente, EventArgs args)
{
    IsChecked = IsChecked!;

}
}

```

Observe la TypeConverter sobre el Tamaño de fuente propiedad. Eso permite que la propiedad que se encuentra en XAML con valores de atributos tales como "pequeño" y "grande".

los aprovechado controlador para el TapGestureRecognizer está en el fondo de la clase y simplemente alterna el Está chequeado propiedad utilizando el C # operador de negación lógica. Una declaración aún más corto para cambiar una variable booleana utiliza el operador OR exclusivo de asignación:

IsChecked ^ = cierto ;

los **CheckBoxDemo** programa es muy similar a la **SwitchDemo** programa excepto que el margen de beneficio se simplifica considerablemente debido a que la Clase incluye su propio **Texto** propiedad:

```
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    xmlns: kit de herramientas =
        "clr-espacio de nombres: Xamarin.FormsBook.Toolkit; montaje = Xamarin.FormsBook.Toolkit"
    x: Class = " CheckBoxDemo.CheckBoxDemoPage " >

< StackLayout Relleno = " 10, 0 " >
    < StackLayout HorizontalOptions = " Centrar "
        VerticalOptions = " CenterAndExpand " >

        < kit de herramientas: CheckBox Texto = " Ítlico "
            Tamaño de fuente = " Grande "
            CheckedChanged = " OnitalicCheckBoxChanged " />

        < kit de herramientas: CheckBox Texto = " negrita "
            Tamaño de fuente = " Grande "
            CheckedChanged = " OnBoldCheckBoxChanged " />

    </ StackLayout >

    < Etiqueta x: Nombre = " etiqueta "
        Texto =
        " Sólo un pequeño paso de texto de ejemplo que se pueden formatear
en cursiva o negrita altermando los dos puntos de vista CheckBox personalizado. "
        Tamaño de fuente = " Grande "
        HorizontalTextAlignment = " Centrar "
        VerticalOptions = " CenterAndExpand " />

</ StackLayout >
```

</ Pagina de contenido >

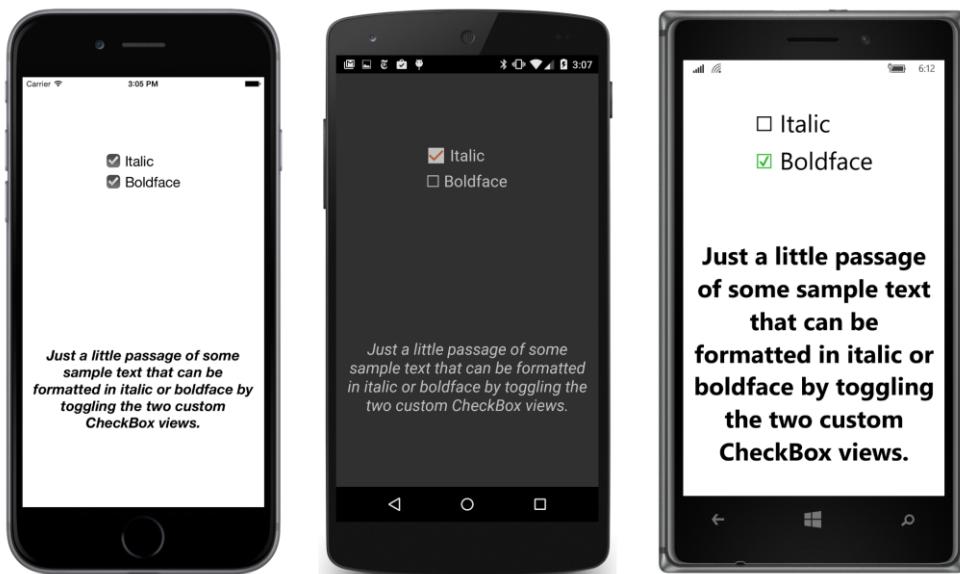
El archivo de código subyacente es también muy similar al programa anterior:

```
pública clase parcial CheckBoxDemoPage : Pagina de contenido
{
    pública CheckBoxDemoPage ()
    {
        InitializeComponent ();
    }

    vacío OnItalicCheckBoxChanged ( objeto remitente, bool está chequeado )
    {
        Si (está chequeado)
        {
            label.FontAttributes |= FontAttributes .Itálico;
        }
        más
        {
            label.FontAttributes &= ~ FontAttributes .Itálico;
        }
    }

    vacío OnBoldCheckBoxChanged ( objeto remitente, bool está chequeado )
    {
        Si (está chequeado)
        {
            label.FontAttributes |= FontAttributes .Negrita;
        }
        más
        {
            label.FontAttributes &= ~ FontAttributes .Negrita;
        }
    }
}
```

Curiosamente, el carácter de la casilla seleccionada aparece en color en las plataformas Android y Windows:



Escribir texto

Xamarin.Forms define tres vistas para la obtención de la introducción de texto por parte del usuario:

- Entrada para una sola línea de texto.
- Editor para múltiples líneas de texto.
- Barra de búsqueda para una sola línea de texto específicamente para las operaciones de búsqueda.

Ambos Entrada y Editor derivarse de InputView, que se deriva de Ver. Barra de búsqueda deriva directamente de Ver.

Ambos Entrada y Barra de búsqueda implementar el desplazamiento horizontal si el texto introducido supera el ancho de la vista. Los Editor implementa ajuste de texto y es capaz de desplazamiento vertical para el texto que excede su altura.

Teclado y el enfoque

Entrada, Editor, y Barra de búsqueda son diferentes de todos los demás puntos de vista en cuanto a que hagan uso de teclado en pantalla del teléfono, a veces llamado el *teclado virtual*. Desde la perspectiva del usuario, tocando el Entrada, Editor, o Barra de búsqueda Ver invoca el teclado en pantalla, que se desliza desde la parte inferior. Al tocar en ningún otro lugar en la pantalla (excepto otro Entrada, Editor, o Barra de búsqueda vista) a menudo hace que desaparezca el teclado y, a veces el teclado puede ser despedido de otras maneras.

Desde la perspectiva del programa, la presencia del teclado está estrechamente relacionado con *foco de entrada*, un concepto que se originó en los entornos de interfaz gráfica de usuario. En ambos entornos de escritorio y dispositivos móviles, entrada desde el teclado puede ser dirigida a un solo objeto de interfaz de usuario a la vez, y ese objeto debe ser claramente seleccionable e identificable por el usuario. El objeto que recibe la entrada de teclado se conoce como el objeto con *de entrada foco del teclado*, o, más sencillamente, simplemente *foco de entrada*

o *atención*.

los VisualElement clase define varios métodos, propiedades y eventos relacionados con el foco de entrada:

- los Atención método intenta establecer foco de entrada a un elemento visual y vuelve cierto si tiene éxito.
- los unfocus método elimina foco de entrada desde un elemento visual.
- los Esta enfocado llegar establecimiento sólo es cierto si un elemento visual tiene actualmente el foco de entrada.
- los centrado evento se activa cuando un elemento visual adquiere el foco de entrada.
- los Desenfocado evento se activa cuando un elemento visual pierde el foco de entrada.

Como ya saben, los entornos móviles hacen menos uso del teclado de los entornos de escritorio hacen, y la mayoría de las vistas móviles (como el Deslizador, paso a paso, y Cambiar que ya ha visto) no haga uso del teclado en absoluto. A pesar de esos cinco miembros relacionados de enfoque-de la VisualElement

clase aparece para poner en práctica un sistema generalizado para pasar el foco de entrada entre los elementos visuales, que en realidad sólo se refieren a Entrada, Editor, y Barra de búsqueda.

Estos puntos de vista señalan de que tienen el foco de entrada con un cursor parpadeante que muestra el punto de entrada de texto, y desencadenar el teclado para deslizarse hacia arriba. Cuando la vista pierde el foco de entrada, el teclado se desliza hacia abajo.

Una visión debe tener su Esta habilitado propiedad establecida en cierto (el estado por defecto) para adquirir el foco de entrada, y por supuesto la Es visible la propiedad debe ser también cierto o la vista no estará en la pantalla en absoluto.

La elección del teclado

Entrada y Editor son diferentes de Barra de búsqueda en el que ambos se derivan de InputView. Curiosamente, a pesar Entrada y Editor definir las propiedades y eventos similares, InputView define una sola propiedad: Teclado. Esta propiedad permite que un programa para seleccionar el tipo de teclado que se muestra. Por ejemplo, un teclado para escribir una dirección URL debe ser diferente de un teclado para introducir un número de teléfono. Las tres plataformas tienen diferentes estilos de teclados virtuales correspondientes para los diferentes tipos de entrada de texto. Un programa no puede seleccionar el teclado utilizado para Barra de búsqueda.

Esta Teclado propiedad es de tipo Teclado, una clase que define siete propiedades de sólo lectura estáticas de tipo Teclado apropiarse para usos diferentes de teclado:

- Defecto
- Texto

- Charla
- url
- Email
- Teléfono
- Numérico

En las tres plataformas, la Numérico teclado permite escribir puntos decimales, pero no permite escribir un signo negativo, por lo que se limita a los números positivos.

El programa siguiente crea siete Entrada vistas que le permiten ver cómo se implementan estos teclados en las tres plataformas. El teclado particular, unido a cada Entrada se identifica por una propiedad definida por Entrada llamado Marcador de posición. Este es el texto que aparece en el Entrada antes de nada que el usuario escribe como una sugerencia para la naturaleza del texto del programa está a la espera. texto del marcador es comúnmente una corta frase como "Nombre" o "Dirección de correo electrónico":

```
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " EntryKeyboards.EntryKeyboardsPage " >

    < ContentPage.Padding >
        < OnPlatform x: TypeArguments = " Espesor "
            iOS = " 10, 20, 10, 0 "
            Androide = " 10, 0 "
            WinPhone = " 10, 0 " />
    </ ContentPage.Padding >

    < ScrollView >
        < StackLayout >
            < StackLayout.Resources >
                < ResourceDictionary >
                    < Estilo Tipo de objetivo = " Entrada " >
                        < Setter Propiedad = " VerticalOptions " Valor = " CenterAndExpand " />
                    </ Estilo >
                </ ResourceDictionary >
            </ StackLayout.Resources >

            < Entrada marcador de posición = " Defecto "
                Teclado = " Defecto " />

            < Entrada marcador de posición = " Texto "
                Teclado = " Texto " />

            < Entrada marcador de posición = " Charla "
                Teclado = " Charla " />

            < Entrada marcador de posición = " url "
                Teclado = " url " />

            < Entrada marcador de posición = " Email "
                Teclado = " Email " />
        </ StackLayout >
    </ ScrollView >
</ Pagina de contenido >
```

```

Teclado = "Email" />

<Entrada marcador de posición = "Teléfono"
  Teclado = "Teléfono" />

<Entrada marcador de posición = "Numérico"
  Teclado = "Numérico" />

</StackLayout>
</ScrollView>
</Página de contenido>

```

Los marcadores de posición aparecen como texto gris. Así es como se ve la pantalla cuando el programa primero comienza a correr:



Al igual que con la deslizador, usted no desea establecer `HorizontalOptions` en una Entrada a Izquierda, Centro, o Derecha a menos que también establece la `WidthRequest` propiedad. Si lo hace, el Entrada colapsa a una anchura muy pequeña. Todavía se puede utilizar el Entrada proporciona automáticamente el desplazamiento horizontal para el texto más largo que el Entrada puede mostrar, pero que realmente debe tratar de proporcionar un tamaño adecuado. En este programa cada Entrada es tan ancha como la pantalla menos un acolchado 10-unidad de la izquierda y la derecha.

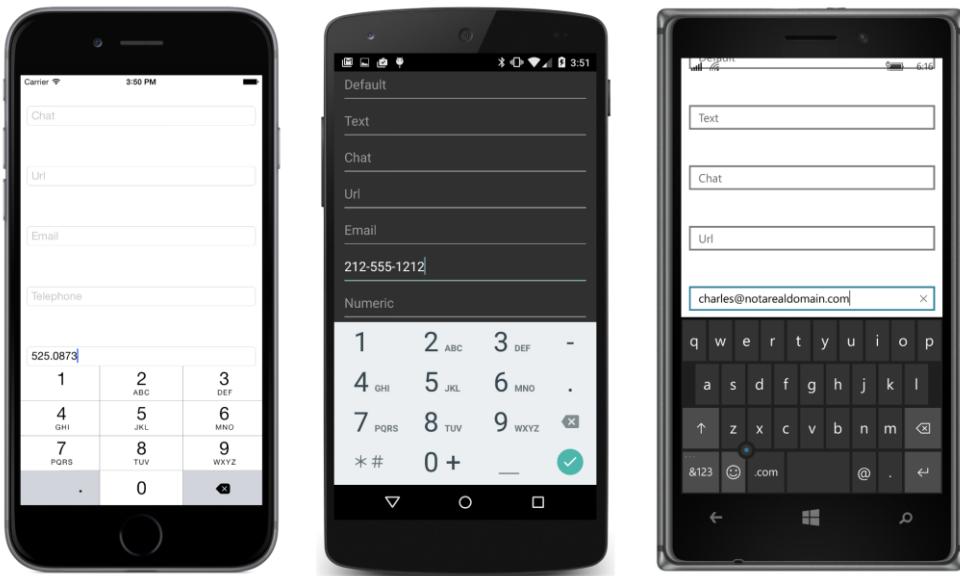
Se puede estimar una adecuada `WidthRequest` través de la experimentación con diferentes longitudes de texto. El próximo programa en este capítulo establece el Entrada anchura a un valor equivalente a una pulgada.

los `EntryKeyboards` programa de manera uniforme espacios del siete Entrada vistas verticalmente utilizando una `VerticalOptions` valor de `CenterAndExpand` establecer a través de un estilo implícita. Es evidente que hay suficiente espacio vertical para los siete Entrada puntos de vista, por lo que podrían ser confundidos acerca de la utilización de la `ScrollView` en el archivo XAML.

los `ScrollView` es específicamente para IOS. Si toca una Entrada cerca de la parte inferior de la Android o

Windows 10 pantalla móvil, el sistema operativo se moverá automáticamente el contenido de la página cuando aparezca el teclado, por lo que la Entrada aún es visible mientras está escribiendo. Pero iOS no lo hace a menos que una ScrollView está provisto.

Así es como se ve cada pantalla cuando el texto está siendo escrito en una de las Entradas vistas hacia la parte inferior de la pantalla:



Propiedades de entrada y eventos

Además de heredar el Teclado propiedad de InputView, Entrada define cuatro propiedades más, de las que solamente se vio en el programa anterior:

- **Texto** - la cadena que aparece en el Entrada
- **Color de texto** - un Color valor
- **IsPassword** - un booleano que hace que los caracteres a ser enmascarado derecha después de que se escriben
- **marcador de posición** - El texto de color claro que aparece en el Entrada pero desaparece tan pronto como el usuario empieza a escribir.

Generalmente, un programa obtiene lo que el usuario escribió accediendo a la Texto propiedad, pero el programa también puede inicializar el Texto propiedad. Tal vez el programa desea sugerir una cierta entrada de texto.

los Entrada también define dos eventos:

- **TextChanged**

- Terminado

los **TextChanged** evento es disparado por cada cambio en el **Texto** propiedad, que generalmente corresponde a cada golpe de teclado (excepto turno y algunas teclas especiales). Un programa puede controlar este evento para realizar comprobaciones de validez. Por ejemplo, es posible comprobar si hay números válidos o direcciones de correo electrónico válidas que permitan una **Calcular** o **Enviar** botón.

Los **Terminado** evento se activa cuando el usuario pulsa una tecla en particular en el teclado para indicar que el texto se ha completado. Esta clave es específica de la plataforma:

- **iOS:** La clave está etiquetado **regreso**, que no está en el Teléfono o Numérico teclado.
- Android: La clave es una marca de verificación verde en la esquina inferior derecha del teclado.
- **Windows Phone:** La clave es una Enter (o retorno) símbolo (↵) en la mayoría de los teclados, pero es un símbolo Go (→) sobre el url teclado. Una tecla de este tipo no está presente en el Teléfono y Numérico teclados.

En iOS y Android, la clave completado descarta el teclado, además de la generación de la completó evento. En Windows 10 móvil no lo hace.

Android y los usuarios de Windows también pueden ocultar el teclado utilizando el teléfono de **Espalda** botón en la parte inferior izquierda de la pantalla vertical. Esto hace que el Entrada perder el foco de entrada, pero no causa la Terminado evento al fuego.

Vamos a escribir un programa llamado **Ecuaciones cuadráticas** que resuelve las ecuaciones cuadráticas, que son ecuaciones de la forma:

$$\bullet \cdot z + \bullet \cdot + \bullet = 0$$

Para cualquier tres constantes *a*, *b*, y *c*, el programa utiliza la ecuación cuadrática para resolver *X*:

$$\bullet = -\frac{b \pm \sqrt{b^2 - 4ac}}{2a}$$

Se entra *a*, *b*, y *c* de cada tres Entrada puntos de vista y pulse una Botón etiquetado **Solución para x**.

Aquí está el archivo XAML. Desafortunadamente, el Numérico teclado no es adecuado para este programa porque en las tres plataformas que no permite la introducción de números negativos. Por esa razón, no se especifica ningún teclado en particular:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
      xmlns:x = " http://schemas.microsoft.com/winfx/2009/xaml "
      x: Class = " QuadraticEquations.QuadraticEquationsPage " >

< ContentPage.Resources >
    < ResourceDictionary >
        < Estilo Tipo de objetivo = " Etiqueta " >
            < Setter Propiedad = " Tamaño de fuente " Valor = " Medio " />
            < Setter Propiedad = " VerticalOptions " Valor = " Centrar " />
        </ Estilo >
    </ ResourceDictionary >
</ ContentPage.Resources >
```

```
< Estilo Tipo de objetivo = " Entrada " >
    < Setter Propiedad = " WidthRequest " Valor = " 180 " />
</ Estilo >
</ ResourceDictionary >
</ ContentPage.Resources >

< StackLayout >
    <l - sección de entrada >
        < StackLayout Relleno = " 20, 0, 0, 0 "
            VerticalOptions = " CenterAndExpand "
            HorizontalOptions = " Centrar " >

            < StackLayout Orientación = " Horizontal " >
                < Entrada x: Nombre = " entryA "
                    TextChanged = " OnEntryTextChanged "
                    Terminado = " OnEntryCompleted " />
                < Etiqueta Texto = " X & # 178; + " />
            </ StackLayout >

            < StackLayout Orientación = " Horizontal " >
                < Entrada x: Nombre = " entryB "
                    TextChanged = " OnEntryTextChanged "
                    Terminado = " OnEntryCompleted " />
                < Etiqueta Texto = " x + " />
            </ StackLayout >

            < StackLayout Orientación = " Horizontal " >
                < Entrada x: Nombre = " entryC "
                    TextChanged = " OnEntryTextChanged "
                    Terminado = " OnEntryCompleted " />
                < Etiqueta Texto = " = 0 " />
            </ StackLayout >
        </ StackLayout >
    <l - Botón >
    < Botón x: Nombre = " solveButton "
        Texto = " Solución para x "
        Tamaño de fuente = " Grande "
        Está habilitado = " Falso "
        VerticalOptions = " CenterAndExpand "
        HorizontalOptions = " Centrar "
        hecho clic = " OnSolveButtonClicked " />

    <l - sección de resultados >
    < StackLayout VerticalOptions = " CenterAndExpand "
        HorizontalOptions = " Centrar " >
        < Etiqueta x: Nombre = " solution1Label "
            HorizontalTextAlignment = " Centrar " />

        < Etiqueta x: Nombre = " solution2Label "
            HorizontalTextAlignment = " Centrar " />
    </ StackLayout >
</ StackLayout >
</ Pagina de contenido >
```

los Etiqueta, de entrada, y Botón las opiniones están divididas en tres secciones: la introducción de datos en la parte superior, la Botón en el medio, y los resultados en la parte inferior. Note la plataforma específica WidthRequest establecer en el implícito Estilo Para el Entrada. Esto le da a cada uno Entrada una anchura de una pulgada.

El programa ofrece dos formas de activar un cálculo: pulsando la tecla de finalización en el teclado, o pulsando el Botón en el centro de la página. Otra opción en un programa como este sería llevar a cabo el cálculo para cada golpe de teclado (o para ser más exactos, cada TextChanged

evento). Esto funcionaría aquí porque el cálculo es muy rápida. Sin embargo, en el presente diseño de los resultados están cerca de la parte inferior de la pantalla y están cubiertos cuando el teclado virtual está activo, por lo que la página tendría que ser reorganizado para tal esquema a tener sentido.

los **Ecuaciones cuadráticas** utiliza el programa TextChanged evento, pero únicamente para determinar la validez del texto escrito en cada Entrada. El texto se pasa a Double.TryParse, y si el método devuelve falso, el Entrada texto se muestra en rojo. (En Windows 10 móvil, la coloración roja de texto aparece sólo cuando el Entrada pierde el foco de entrada.) Además, el Botón se activa sólo si los tres Entrada vistas contienen válida doble valores. Aquí está la primera mitad del archivo de código subyacente que muestra toda la interacción del programa:

```
público clase parcial QuadraticEquationsPage : Página de contenido
{
    público QuadraticEquationsPage ()
    {
        InitializeComponent ();

        // Inicializar vistas de entrada.
        entryA.Text = "1";
        entryB.Text = "-1";
        entryC.Text = "-1";
    }

    vacío OnEntryTextChanged ( objeto remitente, TextChangedEventArgs args )
    {
        // soluciones VACIA.
        solution1Label.Text = "";
        solution2Label.Text = "";

        // color del texto de entrada actual, basado en la validez.
        Entrada entrada = ( Entrada )remitente;
        doble resultado;
        entry.TextColor = Doble .TryParse ( entry.Text, fuera resultado )? Color .Defecto : Color .Rojo;

        // Habilitar el botón sobre la base de validez.
        solveButton.IsEnabled = Doble .TryParse ( entryA.Text, fuera resultado ) &&
            Doble .TryParse ( entryB.Text, fuera resultado ) &&
            Doble .TryParse ( entryC.Text, fuera resultado );
    }

    vacío OnEntryCompleted ( objeto remitente, EventArgs args )
    {
        Si ( solveButton.IsEnabled )
        {
            Resolver();
        }
    }
}
```

```

        }

    }

    vacio OnSolveButtonClicked (objeto remitente, EventArgs args)
    {
        Resolver();
    }
    ...
}

```

los Terminado controlador para el Entrada llama a la Resolver método sólo cuando el Botón está habilitada, que (como hemos visto) indica que los tres Entrada vistas contienen valores válidos. Por lo tanto, la Resolver método puede asumir con seguridad que los tres Entrada vistas contienen números válidos que no causarán duplicó con ble.Parse para elevar una excepción.

los Resolver método es necesariamente complicada porque la ecuación cuadrática puede tener uno o dos soluciones, y cada solución podría tener una parte imaginaria, así como una parte real. El método inicializa la parte real de la segunda solución a Double.NaN ("No es un número") y muestra el segundo resultado sólo si ese ya no es el caso. Las partes imaginarias sólo se muestran si son distintos de cero, y, o bien un signo más o un guión (Unicode \ u2013) conecta las partes real e imaginaria:

```

pùblico clase parcial QuadraticEquationsPage : Pagina de contenido
{
    ...
    vacio Resolver()
    {
        doble a = Doble .Parse (entryA.Text);
        doble b = Doble .Parse (entryB.Text);
        doble c = Doble .Parse (entryC.Text);
        doble solution1Real = 0;
        doble solution1Imag = 0;
        doble solution2Real = Doble .Yaya;
        doble solution2Imag = 0;
        cuerda str1 = "";
        cuerda str2 = "";

        Si (A == 0 && b == 0 && c == 0)
        {
            str1 = "X = nada";
        }
        else if (A == 0 && b == 0)
        {
            str1 = "X = nada";
        }
        más
        {
            Si (A == 0)
            {
                solution1Real = -c / b;
            }
            más
            {

```

```

doble discriminante = b * b - 4 * a * c;

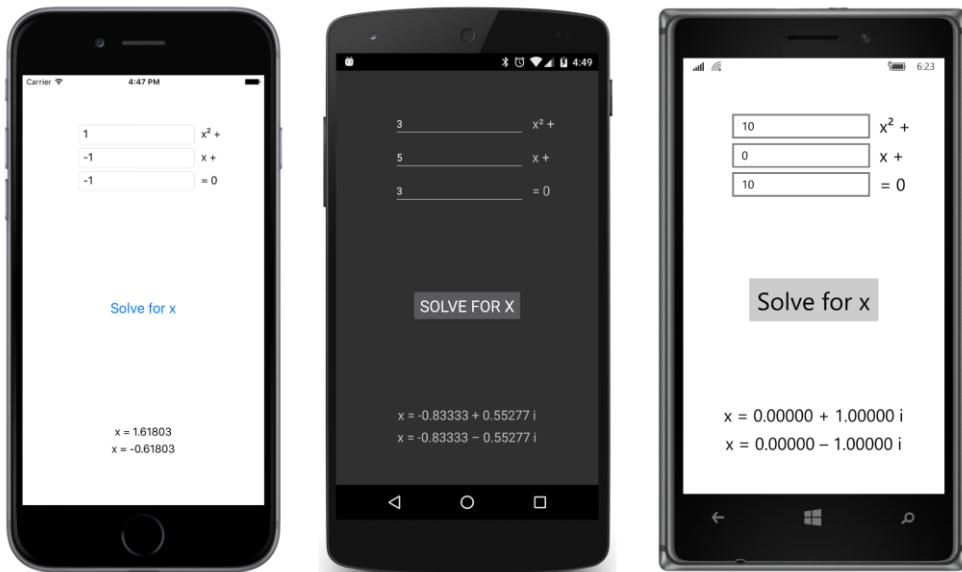
Si (== discriminante 0)
{
    solution1Real = -b / (2 * a);
}
else if (Discriminante> 0)
{
    solution1Real = (-b + Mates .Sqrt (discriminante)) / (2 * a);
    solution2Real = (-b - Mates .Sqrt (discriminante)) / (2 * a);
}
más
{
    solution1Real = -b / (2 * a);
    solution2Real = solution1Real;

    solution1Imag = Mates .Sqrt (-discriminant) / (2 * a);
    solution2Imag = -solution1Imag;
}
}
str1 = Format (solution1Real, solution1Imag);
str2 = Format (solution2Real, solution2Imag);
}
solution1Label.Text = str1;
solution2Label.Text = str2;
}

cuerda Formato( doble real, doble imag)
{
    cuerda str = "";
    Si (| Doble .IsNaN (real))
    {
        str = Cuerda .Formato( "X = {0: F5}" , Real);
        Si (Imag! = 0)
        {
            str += Cuerda .Formato( "{0} {1: F5} i" ,
                Mates .sign (imag) == 1? "+" : " U2013" ,
                Mates .Abs (imag));
        }
    }
    regreso str;
}
}

```

Aquí hay un par de soluciones:



La diferencia Editor

Es posible suponer que la Editor tiene una más amplia que la API Entrada ya que puede manejar múltiples líneas e incluso párrafos de texto. Pero en Xamarin.Forms, la API para Editor en realidad es algo más sencilla. Además de heredar el Teclado propiedad de InputView, Editor define una sola propiedad en su propia: lo esencial Texto propiedad. Editor también define los mismos dos eventos como Entrada:

- TextChanged
- Terminado

sin embargo, el Terminado caso de necesidad es un poco diferente. Mientras que una tecla Intro puede ser señal de finalización en una Entrada, estas mismas teclas utilizadas con el Editor en vez marcar el final de un párrafo.

los Terminado acontecimiento para Editor funciona de manera diferente en las tres plataformas: Para iOS, Xamarin.Forms muestra una especial **Hecho** botón situado encima del teclado que despieza el teclado y causa una Terminado evento al fuego. En Android y Windows Mobile 10, el sistema de **Espalda** -botón el botón situado en la esquina inferior izquierda del teléfono en modo vertical-despieza el teclado y dispara el Terminado evento. Esta **Espalda** botón hace *no* disparar el Terminado para un evento Entrada ver, pero lo hace ocultar el teclado.

Es probable que lo que los usuarios escriben en una Editor No se números de teléfono y direcciones URL reales, pero las palabras, frases y párrafos. En la mayoría de los casos, tendrá que utilizar el Texto teclado para Editor ya que proporciona controles de ortografía, sugerencias y automática en mayúsculas de la primera palabra de las oraciones. Si no desea que estas características, el Teclado clase proporciona un medio alternativo de la especificación de un teclado mediante el uso de un estático Crear método y los siguientes miembros de la KeyboardFlags enumeración:

- CapitalizeSentence (igual a 1)
- Corrector ortográfico (2)
- sugerencias (4)
- Todas (\xFFFFFFFF)

los Texto teclado es equivalente a la creación del teclado con KeyboardFlags.All. los Defecto

teclado es equivalente a la creación del teclado con (KeyboardFlags) 0. No se puede crear un teclado en XAML utilizando estas banderas.
Debe ser hecho en código.

los JustNotes programa pretende ser un programa para tomar notas de forma libre que guarda y restaura el contenido de una forma automática Editor vista mediante el uso de la propiedades colección de la Solicitud clase. La página, básicamente, consiste en un gran Editor, pero para dar al usuario alguna pista acerca de lo que hace el programa, el nombre del programa se muestra en la parte superior. En iOS y Android, tales texto puede ser establecido por el Título propiedad de la página, sino para mostrar que la propiedad, la Pagina de contenido debe ser envuelto en una ApplicationPage (como has descubierto con el ToolbarDemo programa en el Capítulo 13). Eso se hace en el constructor de la Aplicación clase:

```
clase pública Aplicación : Solicitud
{
    público App ()
    {
        MainPage = nuevo NavigationPage ( nuevo JustNotesPage ());
    }

    protegido override void OnStart ()
    {
        // manipulador cuando se inicia el app
    }

    protegido override void OnSleep ()
    {
        // manipulador cuando su aplicación duerme
        (( JustNotesPage ) ((( NavigationPage ) MainPage ) .CurrentPage)) .OnSleep ();
    }

    protegido override void En resumen()
    {
        // manejar cuando sus hojas de vida de aplicaciones
    }
}
```

los OnSleep método en el Aplicación llama a un método llamado OnSleep se define en la JustNotesPage archivo de código subyacente. Así es como el contenido de la Editor se guardan en la memoria de la aplicación.

El elemento raíz de la página XAML establece el Título propiedad. El resto de la página está ocupada por una AbsoluteLayout lleno del Editor:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
```

```

    xmlns:x = "http://schemas.microsoft.com/winfx/2009/xaml"
    x: Class = "JustNotes.JustNotesPage"
    Título = "sólo Notas"

    < StackLayout >
        < AbsoluteLayout VerticalOptions = "FillAndExpand" >
            < Editor x: Nombre = "editor"
                Teclado = "Texto"
                AbsoluteLayout.LayoutBounds = "0, 0, 1, 1"
                AbsoluteLayout.LayoutFlags = "Todas"
                centrado = "OnEditorFocused"
                Desenfocado = "OnEditorUnfocused" />
        </ AbsoluteLayout >
    </ StackLayout >
</ Pagina de contenido >

```

Entonces, ¿por qué usar el programa de una `AbsoluteLayout` para acoger la `Editor`?

los **JustNotes** programa es un trabajo en progreso. Lo hace no está bien el trabajo para iOS. Como se recordará, cuando una Entrada punto de vista se coloca hacia la parte inferior de la pantalla, que desea poner en una Volata-Ver por lo que desplaza hacia arriba cuando se visualiza el teclado virtual IOS. Sin embargo, debido Editor implementa su propio desplazamiento, no se puede poner en una ScrollView.

Por esa razón, el archivo de código subyacente define la altura de la Editor a la mitad de la altura de la `AbsoluteLayout` cuando el Editor recibe el foco de entrada para que el teclado no se solapa con ella, y restaura la Editor altura cuando pierde el foco de entrada:

```

público clase parcial JustNotesPage : Pagina de contenido
{
    público JustNotesPage ()
    {
        InitializeComponent ();

        // Obtiene guardó por última vez el texto Editor.
        IDictionary < cuenda , objeto > properties = Solicitud .Current.Properties;

        Si (Properties.ContainsKey ( "texto" ))
        {
            editor.Text = ( cuenda ) propiedades [ "texto" ];
        }
    }

    vacío OnEditorFocused ( objeto remitente, FocusEventArgs args)
    {
        Si ( Dispositivo .os == TargetPlatform .iOS)
        {
            AbsoluteLayout .SetLayoutBounds (editor, nuevo Rectángulo (0, 0, 1, 0,5));
        }
    }

    vacío OnEditorUnfocused ( objeto remitente, FocusEventArgs args)
    {
        Si ( Dispositivo .os == TargetPlatform .iOS)

```

```

    {
        AbsoluteLayout.SetLayoutBounds(editor, nuevo Rectángulo (0, 0, 1, 1));
    }
}

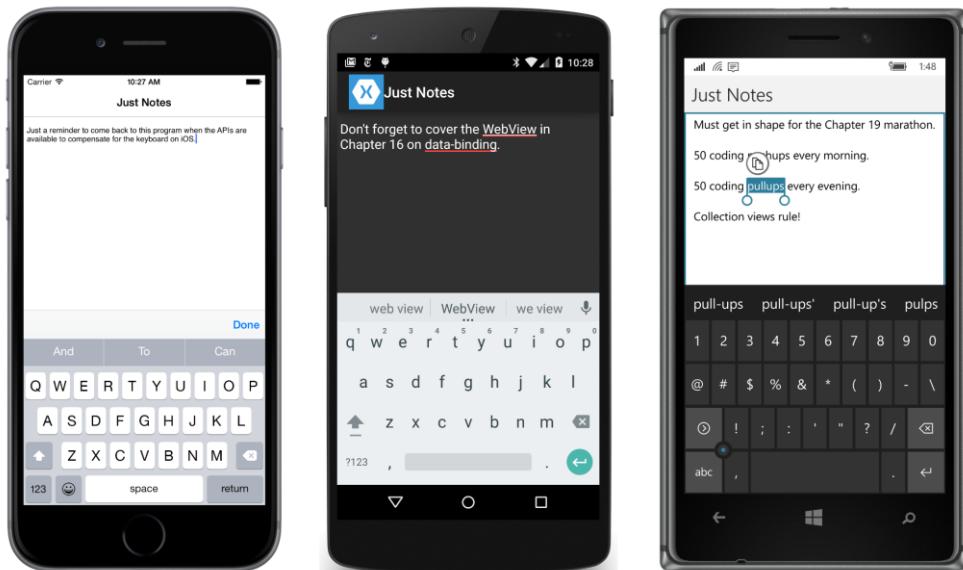
public void OnSleep ()
{
    // Guardar el texto Editor.
    Solicitud.Current.Properties ["texto"] = Editor.Text;
}
}

```

Ese ajuste es sólo aproximada, por supuesto. Varía según el dispositivo, y que varía según el modo vertical y horizontal, pero suficiente información no está disponible actualmente en Xamarin.Forms hacerlo con mayor precisión. Por ahora, probablemente debería restringir su uso de Editor vistas a la zona superior de la página.

El código para guardar y restaurar la Editor el contenido es bastante prosaico en comparación con el Editor manipulación. los OnSleep método (llamado a partir de la Aplicación clase) guarda el texto en el propiedades diccionario con una tecla de “texto” y el constructor restaura.

Aquí está el programa que se ejecuta en las tres plataformas con la Texto Teclado a la vista con sugerencias de palabras. En la pantalla de Windows Mobile 10, una palabra se ha seleccionado y puede ser copiado en el portapapeles para pegar elementos más tarde:



el SearchBar

los Barra de búsqueda no se deriva de InputView me gusta Entrada y Editor, y que no tiene una Llave-

tablero propiedad. El teclado que Barra de búsqueda Aparece cuando se adquiere el foco de entrada es una plataforma específica y apropiada para un comando de búsqueda. los Barra de búsqueda sí es similar a una Entrada punto de vista, pero dependiendo de la plataforma, que podría ser adornado con algunos otros gráficos y contiene un botón que borra el texto escrito.

Barra de búsqueda define dos eventos:

- TextChanged
- SearchButtonPressed

los TextChanged evento permite a su programa para acceder a una entrada de texto en curso. Tal vez su programa realmente puede comenzar una búsqueda u ofrecer sugerencias específicas del contexto antes de que el usuario lleva a cabo a escribir. los

SearchButtonPressed evento es equivalente a la Terminado disparada por evento Entrada. Se activa mediante un botón especial en el teclado en la misma ubicación que el botón para completar Entrada pero posiblemente marcado de manera diferente.

Barra de búsqueda define cinco propiedades:

- Texto - el texto introducido por el usuario
- marcador de posición - texto de sugerencia aparece antes de que el usuario comienza a escribir
- CancelButtonColor - Tipo de Color
- SearchCommand - para su uso con el enlace de datos
- SearchCommandParameter - para su uso con el enlace de datos

los **SearchBarDemo** programa utiliza solamente Texto y marcador de posición, pero el archivo XAML se une controladores para ambos eventos:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns:x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " SearchBarDemo.SearchBarDemoPage " >

< ContentPage.Padding >
    < OnPlatform x: TypeArguments = " Espesor "
        iOS = " 10, 20, 10, 0 "
        Androide = " 10, 0 "
        WinPhone = " 10, 0 " />
</ ContentPage.Padding >

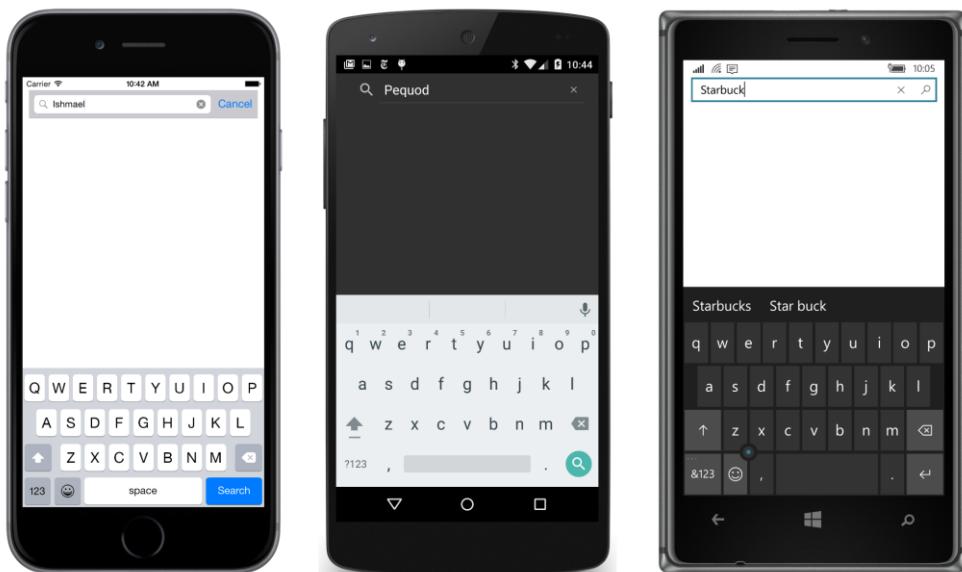
< StackLayout >
    < Barra de búsqueda x: Nombre = " barra de búsqueda "
        marcador de posición = " Buscar texto "
        TextChanged = " OnSearchBarTextChanged "
        SearchButtonPressed = " OnSearchBarButtonPressed " />

    < ScrollView x: Nombre = " resultsScroll "
        VerticalOptions = " FillAndExpand " >
        < StackLayout x: Nombre = " resultsStack " />
    </ ScrollView >
```

```
</ StackLayout >
</ Pagina de contenido >
```

El programa utiliza el desplazable StackLayout llamado resultsStack para mostrar los resultados de la búsqueda.

Aquí está la Barra de búsqueda y el teclado para las tres plataformas. Observe el ícono de búsqueda y un botón de eliminar en las tres plataformas, y las claves de búsqueda especiales en los teclados de iOS y Android:



Se puede adivinar por las entradas en las tres Barra de búsqueda considera que el programa permite buscar a través del texto de Herman Melville *Moby Dick*. ¡Eso es verdad! Toda la novela se almacena como un recurso incrustado en el **Textos** carpeta del proyecto de biblioteca de clases portátil con el nombre *MobyDick.txt*. El archivo es un formato de texto plano, una línea-por-párrafo que se originó con un archivo en el sitio web Gutenberg.net.

El constructor del archivo de código subyacente que lee todo el archivo en un campo de cadena nombrado *bookText*. Los *TextChanged* despeja el manejador *resultsStack* de los resultados de búsqueda anteriores, de modo que no hay discrepancia entre el texto que se escribe en el Barra de búsqueda y esta lista. Los *SearchButton*-Presionado evento inicia la búsqueda:

```
pública clase parcial SearchBarDemoPage : Pagina de contenido
{
    const doble MaxMatches = 100;
    cuerda bookText;

    pública SearchBarDemoPage ()
    {
        InitializeComponent ();
    }
}
```

```
// Cargar incrustado de mapa de bits de recursos.
cuerda de resourceId = "SearchBarDemo.Texts.MobyDick.txt";
Montaje montaje = GetType () GetTypeInfo () Asamblea.;

utilizando ( Corriente flujo = assembly.GetManifestResourceStream (de resourceId))
{
    utilizando ( StreamReader lector = nuevo StreamReader (corriente))
    {
        bookText = reader.ReadToEnd ();
    }
}

vacío OnSearchBarTextChanged ( objeto remitente, TextChangedEventArgs args)
{
    resultsStack.Children.Clear ();
}

vacío OnSearchBarButtonPressed ( objeto remitente, EventArgs args)
{
    // Separar resultsStack de diseño.
    resultsScroll.Content = nulo ;

    resultsStack.Children.Clear ();
    SearchBookForText (searchBar.Text);

    // Volver a colocar resultsStack a disposición.
    resultsScroll.Content = resultsStack;
}

vacío SearchBookForText ( cuerda buscar texto)
{
    En t count = 0;
    bool isTruncated = falso ;

    utilizando ( StringReader lector = nuevo StringReader (BookText))
    {
        En t lineNumber = 0;
        cuerda línea;

        mientras ( nulo != (Línea = reader.ReadLine ()))
        {
            lineNumber++;
            En t index = 0;

            mientras (-1 != (Índice = (line.IndexOf (searchText, índice,
StringComparison.OrdinalIgnoreCase))))
            {
                Si (Count == MaxMatches)
                {
                    isTruncated = cierto ;
                    descanso ;
                }
                índice += 1;
            }
        }
    }
}
```

```

// Se añade la información a la StackLayout.
resultsStack.Children.Add (
    nuevo Etiqueta
    {
        text = Cuerda .Formato( "Se ha encontrado en la línea {0}, desplazamiento {1}" ,
            lineNumber, index)
    });

    contar++;
}

Si (IsTruncated)
{
    descanso ;
}
}

// Añadir recuento final a la StackLayout.
resultsStack.Children.Add (
    nuevo Etiqueta
    {
        text = Cuerda .Formato( "{0} {1} partido encontró {2}" ,
            contar,
            contar == 1? "" : "es",
            IsTruncated? " - detenido" : "" )
    });
}
}

```

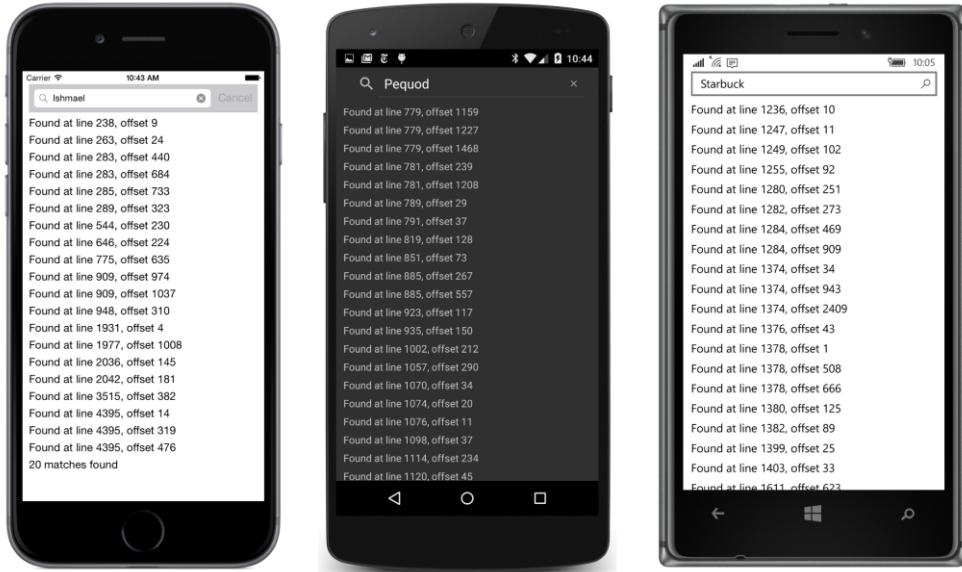
los SearchBookForText método utiliza el texto de búsqueda con el índice de método aplicado para cada línea del libro para la comparación entre mayúsculas y minúsculas y añade una Etiqueta a resultsStack para cada partido. Sin embargo, este proceso tiene problemas de rendimiento, ya que cada Etiqueta que se añade a la StackLayout fuera potencialmente desencadena un nuevo cálculo del esquema. Eso es innecesario. Por esta razón, antes de comenzar la búsqueda, el programa separa la StackLayout Del árbol visual mediante el establecimiento de la Contenido propiedad de su padre (el ScrollView) a nulo:

```
resultsScroll.Content = nulo ;
```

Después de todo el Etiqueta puntos de vista han sido añadidos a la StackLayout, el StackLayout se añade de nuevo al árbol visual:

```
resultsScroll.Content = resultsStack;
```

Pero incluso eso no es una mejora del rendimiento suficiente para algunas búsquedas, y es por eso que el programa se limita a los primeros 100 partidos. (Observe la MaxMatches constante definida en la parte superior de la clase) Aquí está el programa muestra los resultados de las búsquedas que viste escribió anteriormente.:



Tendrá que hacer referencia al archivo real para ver lo que esos partidos son.

Sería ejecutar la búsqueda en un segundo hilo de acelerar las cosas hasta la ejecución? No. La búsqueda de texto real es muy rápido.

Los problemas de rendimiento implican la interfaz de usuario. Si el `SearchBookForText`

método se ejecuta en un subproceso secundario, entonces se tendría que usar `Device.BeginInvokeOnMain-`

Hilo para agregar cada Etiqueta al StackLayout. Si eso StackLayout está unido al árbol visual, esto haría que el programa funcione más dinámicamente los elementos individuales aparecerían en la pantalla después de cada elemento añadido a la lista, pero la conmutación de ida y vuelta entre los hilos retrasaría la operación global.

Fecha y hora de selección

Una aplicación Xamarin.Forms que necesita una fecha o una hora desde el usuario utilice este Selector de fechas o `TimePicker` ver.

Estos son muy similares: los dos puntos de vista simplemente mostrar una fecha u hora en una caja similar a una Entrada ver. Al tocar la vista invoca el selector de fecha u hora específica de la plataforma. El usuario selecciona entonces (o diales en) una nueva fecha o el tiempo y señales de terminación.

DatePicker

Selector de fechas tiene tres propiedades de tipo Fecha y hora:

- `MinimumDate`, inicializado 1 de enero de 1,9 mil

- `MaximumDate`, inicializado al 31 de diciembre 2,100
- `Fecha`, inicializado a `DateTime.Today`

Un programa puede establecer estas propiedades a lo que quiera, siempre y cuando `MinimumDate` es anterior a `MaximumDate`. La `Fecha` propiedad refleja la selección del usuario.

Si desea establecer esas propiedades en XAML, puede hacerlo a través de la `x: DateTime` elemento. Utilice un formato que sea aceptable para el `DateTime.Parse` método con un segundo argumento de `CultureInfo`.

`Info.InvariantCulture`. Probablemente el más fácil es el formato de fecha corta, que es un mes de dos dígitos, un día de dos dígitos, y un año de cuatro dígitos, separados por barras:

```
< Selector de fechas ... >
  < DatePicker.MinimumDate >
    03/01/2016
  </ DatePicker.MinimumDate >

  < DatePicker.MaximumDate >
    31/10/2016
  </ DatePicker.MaximumDate >

  < DatePicker.Date >
    04/24/2016
  </ DatePicker.Date >
</ Selector de fechas >
```

Los Selector de fechas muestra la fecha seleccionada mediante el uso de la normal Encadenar método, pero se puede establecer el Formato propiedad de la vista a una cadena de formato personalizada .NET. El valor inicial es "d", el formato de shortdate.

Aquí está el archivo XAML de un programa llamado **DaysBetweenDates** que le permite seleccionar dos fechas y luego se calcula el número de días entre ellos. Contiene dos Selector de fechas vistas etiquetado A y

De:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
  xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
  x: Class = " DaysBetweenDates.DaysBetweenDatesPage " >

  < ContentPage.Padding >
    < OnPlatform x: TypeArguments = " Espesor "
      iOS = " 10, 30, 10, 0 "
      Androide = " 10, 10, 10, 0 "
      WinPhone = " 10, 10, 10, 0 " />
  </ ContentPage.Padding >

  < StackLayout >
    < StackLayout.Resources >
      < ResourceDictionary >
        < Estilo Tipo de objetivo = " Selector de fechas " >
          < Setter Propiedad = " Formato " Valor = " re " />
          < Setter Propiedad = " VerticalOptions " Valor = " Centro " />
          < Setter Propiedad = " HorizontalOptions " Valor = " FillAndExpand " />
```

```

</ Estilo >
</ ResourceDictionary >
</ StackLayout.Resources >

<!-- cabecera texto subrayado -->
< StackLayout Grid.Row = "0" Grid.Column = "0" Grid.ColumnSpan = "2" 
    VerticalOptions = "CenterAndExpand" 
    HorizontalOptions = "Centrar" >
    < Etiqueta Texto = "Días entre fechas" 
        Tamaño de fuente = "Grande" 
        FontAttributes = "Negrita" 
        Color de texto = "Acento" />
    < BoxView Color = "Acento" 
        HeightRequest = "3" />
</ StackLayout >

< StackLayout Orientación = "Horizontal" 
    VerticalOptions = "CenterAndExpand" >
    < Etiqueta Texto = "De:" 
        VerticalOptions = "Centrar" />

    < Selector de fechas x: Nombre = "fromDatePicker" 
        DateSelected = "OnDateSelected" />
</ StackLayout >

< StackLayout Orientación = "Horizontal" 
    VerticalOptions = "CenterAndExpand" >
    < Etiqueta Texto = "A:" 
        VerticalOptions = "Centrar" />

    < Selector de fechas x: Nombre = "toDatePickerController" 
        DateSelected = "OnDateSelected" />
</ StackLayout >

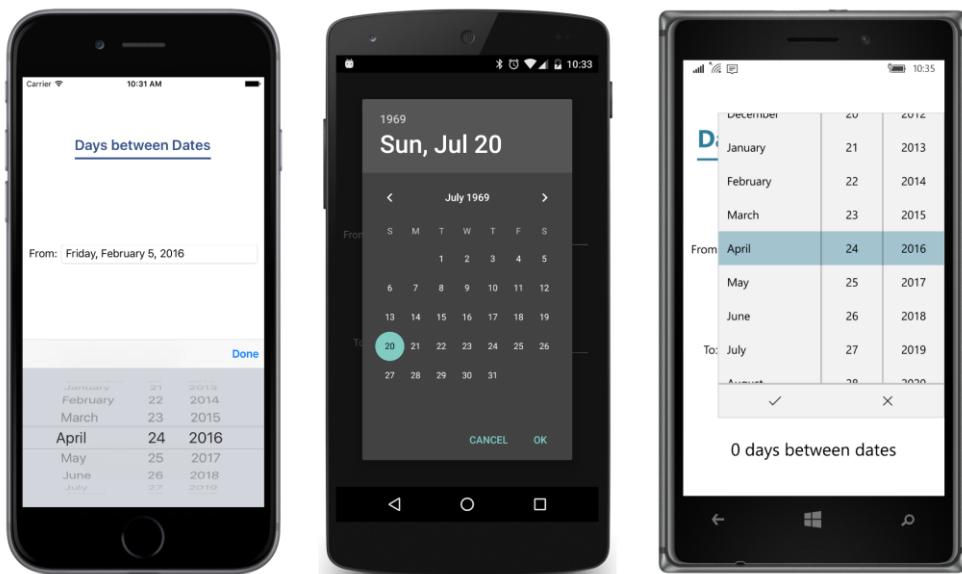
< Etiqueta x: Nombre = "resultado_label" 
    Tamaño de fuente = "Medio" 
    HorizontalOptions = "Centrar" 
    VerticalOptions = "CenterAndExpand" />
</ StackLayout >
</ Pagina de contenido >

```

Un estilo implícita establece el Formato propiedad de los dos Selector de fechas vistas a "D", que es el formato longdate, que incluyen el día de texto de la semana y el mes nombre. El archivo XAML utiliza dos horizontales StackLayout objetos para mostrar una Etiqueta y Selector de fechas lado a lado.

Tenga cuidado: si utiliza el formato de larga fecha, usted querrá evitar el establecimiento de la HorizontalOptions propiedad de la Selector de fechas a Inicio, Centro, o Fin. Si ponemos la Selector de fechas en una horizontal StackLayout (como en este programa), establecer el HorizontalOptions a FillAndExpand. De lo contrario, si el usuario selecciona una cita con una cadena de texto más largo que la fecha original, el resultado no está formateado así. los DaysBetweenDates programa utiliza un estilo implícita para dar la Selector de fechas un HorizontalOptions valor de FillAndExpand de manera que ocupa todo el ancho de la horizontal StackLayout a excepción de lo que está ocupada por la Etiqueta.

Cuando puntee en una de las Selector de fechas campos, un panel específico de la plataforma sube. En iOS, que ocupa sólo la parte inferior de la pantalla, pero en Android y Windows Mobile 10, que más o menos se hace cargo de la pantalla:



Observe la **Hecho** botón en iOS, la **DE ACUERDO** botón en Android, y el botón de barra de herramientas de marca de verificación en Windows Phone. Los tres de estos botones despiden el panel de fecha-picking y volver al programa con un disparo de la `DateSelected` evento. El controlador de eventos en el `DaysBetweenDates` archivo de código subyacente accede tanto Selector de fechas puntos de vista y calcula el número de días entre las dos fechas:

```

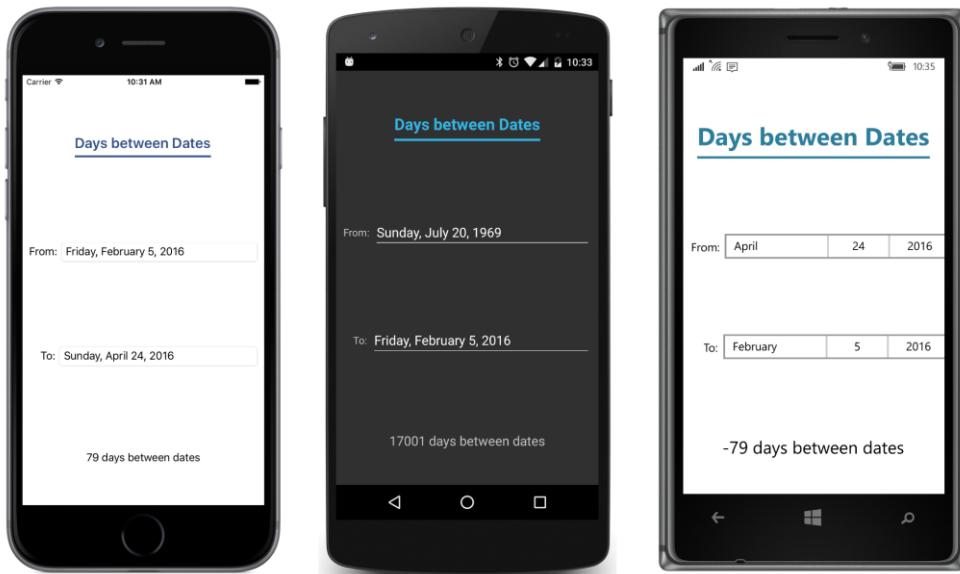
pública clase parcial DaysBetweenDatesPage : Pagina de contenido
{
    pública DaysBetweenDatesPage ()
    {
        InitializeComponent ();

        // Inicializar.
        OnDateSelected ( nulo , nulo );
    }

    vacío OnDateSelected ( objeto remitente, DateChangedEventArgs args )
    {
        En t día = (toDatePicker.Date - fromDatePicker.Date) .Days;
        resultLabel.Text = Cuerta .Formato( "{0} {1} días entre las fechas" ,
            día, día == 1? "" : "S" );
    }
}

```

Aquí está el resultado:



El TimePicker (o es una TimeSpanPicker?)

los TimePicker es algo más sencillo que Selector de fechas. Sólo se define Hora y Formato propiedades, y que no incluye un evento para indicar una nueva seleccionado Hora valor. Si tiene que ser notificado, se puede instalar un controlador para el PropertyChanged evento.

A pesar de que TimePicker muestra el tiempo seleccionado mediante el uso de la Encadenar método de Fecha y hora, el Hora propiedad es en realidad el tipo de Espacio de tiempo, lo que indica una duración de tiempo desde la medianoche.

los SetTimer programa incluye una TimePicker. El programa supone que el tiempo escogido de la TimePicker está dentro de las próximas 24 horas y luego se le notifica cuando ha llegado ese momento. El archivo XAML pone una TimePicker, un Cambiar, y un Entrada en la pagina.

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " SetTimer.SetTimerPage "
    Relleno = " 50 " >

    < StackLayout Espaciado = " 20 "
        VerticalOptions = " Centrar " >
        < TimePicker x: Nombre = " TimePicker "
            PropertyChanged = " OnTimePickerPropertyChanged " />

        < Cambiar x: Nombre = " cambiar "
            HorizontalOptions = " Fin "
            toggled = " OnSwitchToggled " />

        < Entrada x: Nombre = " entrada "
            Texto = " temporizador de muestras " >
```

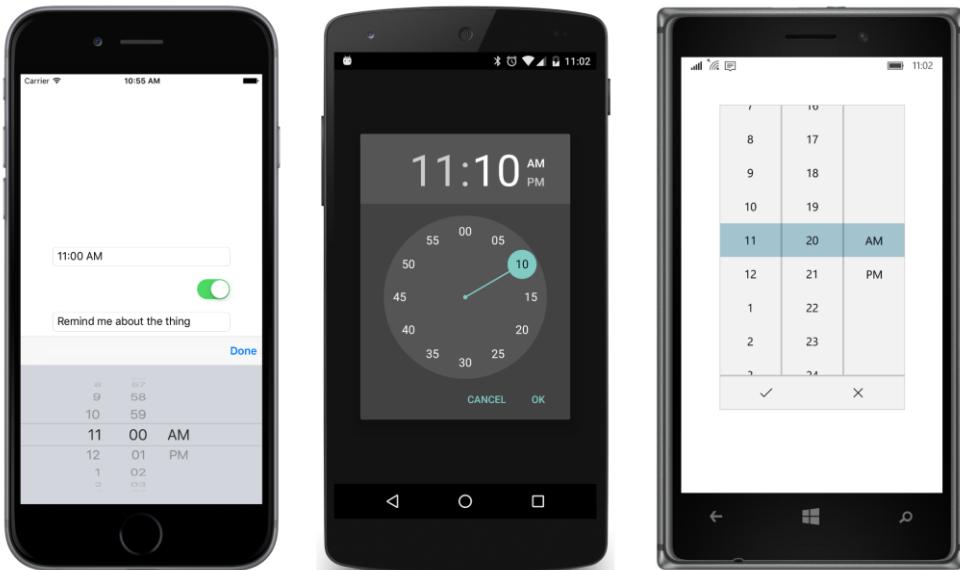
```

    marcador de posición = "etiqueta" />
</StackLayout>
</Página de contenido>

```

los TimePicker tiene un PropertyChanged controlador de eventos adjunta. los Entrada permite usted se recuerda lo que se supone que el temporizador para recordarle.

Al tocar el TimePicker, un panel específico de la plataforma aparece. Al igual que con la Selector de fechas, los 10 paneles móviles Android y Windows ocultan gran parte de la pantalla debajo, pero se puede ver la SetTimer interfaz de usuario en el centro de la pantalla del iPhone:



En un programa de un programa de temporizador real de temporizador que es realmente útil y no sólo una demostración de la TimePicker vista en el archivo de código subyacente podría acceder a las interfaces de notificación específicas de la plataforma para que el usuario sería notificado incluso si el programa ya no estaban activos.

SetTimer no hace eso. **SetTimer** en su lugar utiliza un cuadro de alerta específico de la plataforma que un programa puede invocar mediante una llamada al **DisplayAlert** método que se define por Página y heredado por Pagina de contenido.

los **SetTriggerTime** método en la parte inferior del archivo de código subyacente (que se muestra a continuación) calcula el tiempo del temporizador basado en **DateTime.Today** propiedad -a que devuelve una Fecha y hora indicando la fecha actual, pero con un tiempo de la medianoche, y el Espacio de tiempo regresar de la TimePicker. Si ese tiempo ya ha pasado hoy, entonces se asume que ser mañana.

El contador de tiempo, sin embargo, se establece durante un segundo. Cada segundo de los controles de controlador de temporizador si el Cambiar está encendido y si la hora actual es mayor o igual al tiempo de temporizador:

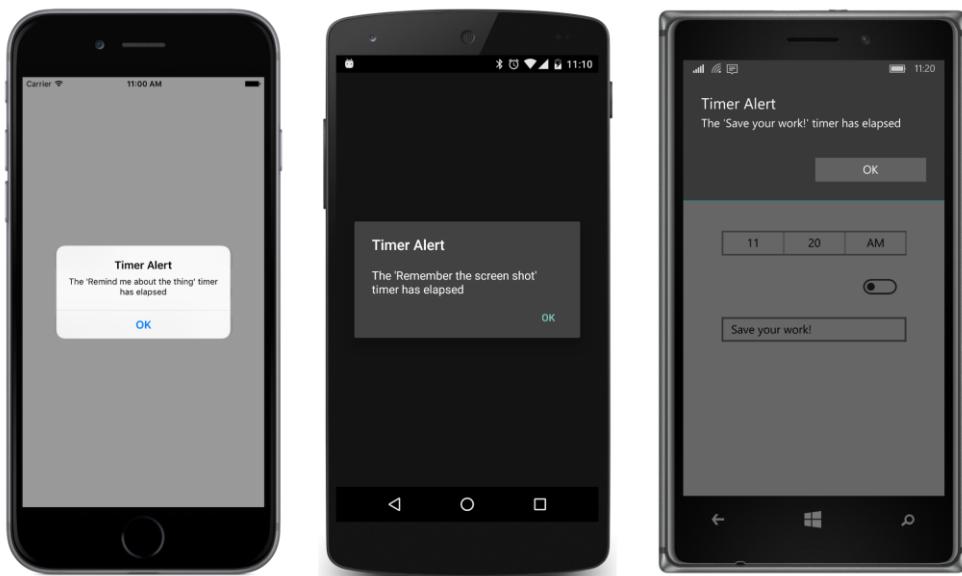
```

pública clase parcial SetTimerPage : Página de contenido
{
    Fecha y hora TriggerTime;
}

```

```
público SetTimerPage ()  
{  
    InitializeComponent ();  
  
    Dispositivo .StartTimer ( Espacio de tiempo .FromSeconds (1), OnTimerTick);  
}  
  
bool OnTimerTick ()  
{  
    Si (@ Switch.IsToggled && Fecha y hora .Now >= TriggerTime)  
    {  
        @ Switch.IsToggled = falso ;  
        DisplayAlert ( "Alerta temporizador" ,  
                      "Los " + + Entry.Text " Temporizador ha transcurrido" ,  
                      "DE ACUERDO" );  
    }  
    return true ;  
}  
  
vacío OnTimePickerPropertyChanged ( objeto obj, PropertyChangedEventArgs args)  
{  
    Si (Args.PropertyName == "Hora")  
    {  
        SetTriggerTime ();  
    }  
}  
  
vacío OnSwitchToggled ( objeto obj, ToggledEventArgs args)  
{  
    SetTriggerTime ();  
}  
  
vacío SetTriggerTime ()  
{  
    Si (@ Switch.IsToggled)  
    {  
        TriggerTime = Fecha y hora .HOY + timePicker.Time;  
  
        Si (TriggerTime < Fecha y hora .Ahora)  
        {  
            TriggerTime += Espacio de tiempo .FromDays (1);  
        }  
    }  
}
```

Cuando ha llegado el tiempo del temporizador, el programa utiliza `DisplayAlert` para señalar un recordatorio para el usuario. Así es como aparece esta alerta en las tres plataformas:



A lo largo de este capítulo, usted ha visto vistas interactivas que definen los acontecimientos, y que ha visto programas de aplicación que implementan controladores de eventos. A menudo, estos controladores de eventos de acceso a una propiedad de la vista y establecer una propiedad de otro punto de vista.

En el siguiente capítulo, verá cómo estos controladores de eventos pueden ser eliminados y cómo las propiedades de los diferentes puntos de vista pueden ser vinculados, ya sea en código o marcado. Esta es la característica interesante de *el enlace de datos*.

capítulo 16

El enlace de datos

Eventos y controladores de eventos son una parte vital de la interfaz interactiva de Xamarin.Forms, pero a menudo los controladores de eventos realizan trabajos muy rudimentarios. Que la transferencia de valores entre las propiedades de diferentes objetos y, en algunos casos, simplemente actualizar una Etiqueta para mostrar el nuevo valor de vistas.

Puede automatizar este tipo de conexiones entre las propiedades de dos objetos con una potente función de llamada *Xamarin.Forms el enlace de datos*. Bajo las sábanas, un enlace de datos instala controladores de eventos y se ocupa de la transferencia de los valores de una propiedad a otra de modo que usted no tenga que hacerlo. En la mayoría de los casos se definen estos enlaces de datos en el archivo XAML, así que no hay código (o muy poco código) que participan. El uso de enlaces de datos ayuda a reducir el número de "partes móviles" en la aplicación.

Los enlaces de datos también juegan un papel crucial en la arquitectura de la aplicación Model-View-ViewModel (MVVM). Como se verá en el capítulo 18, "MVVM," enlaces de datos proporciona el enlace entre la vista (la interfaz de usuario se suelen aplicar en XAML) y los datos subyacentes del modelo de vista y modelo. Esto significa que las conexiones entre la interfaz de usuario y los datos subyacentes pueden ser representados en XAML junto con la interfaz de usuario.

fundamentos vinculantes

Varias propiedades, métodos y clases están implicados en enlaces de datos:

- **los Unión clase (que se deriva de BindingBase) define muchas características de un enlace de datos.**
- **los BindingContext propiedad se define por la bindableObject clase.**
- **los SetBinding método también se define por la bindableObject clase.**
- **los BindableObjectExtensions clase define dos sobrecargas adicionales de SetBinding.**

Dos clases de apoyo extensiones de marcado XAML para los enlaces:

- **los BindingExtension clase, que es privado a Xamarin.Forms, proporciona apoyo a la Unión extensión de marcado que se utiliza para definir un enlace de datos en XAML.**
- **los ReferenceExtension clase también es crucial para fijaciones.**

Dos interfaces también se involucran en el enlace de datos. Estos son:

- **IPropertyChanged (se define en la System.ComponentModel espacio de nombres) es la interfaz estándar que utilizan las clases al notificar clases externas que una propiedad ha cambiado.**

Esta interfaz juega un papel importante en MVVM.

- **IValueConverter** (se define en la `Xamarin.Forms espacio de nombres`) se utiliza para definir clases pequeñas que los datos de ayuda de unión mediante la conversión de los valores de un tipo a otro.

El concepto más fundamental de los enlaces de datos es la siguiente: Los enlaces de datos siempre tienen una *fuente* y una *objetivo*. La fuente es una propiedad de un objeto, por lo general uno que cambia dinámicamente en tiempo de ejecución. Cuando esa propiedad cambia, el enlace de datos se actualicen de forma automática el objetivo, que es una propiedad de otro objeto.

Objetivo ← Fuente

Sin embargo, como se verá, a veces el flujo de datos entre el origen y el destino no está en una dirección constante. Incluso en esos casos, sin embargo, la distinción entre el origen y el destino es importante debido a un hecho básico:

El objetivo de un enlace de datos debe estar respaldada por una BindableProperty objeto.

Como se sabe, el `VisualElement` clase se deriva de `bindableObject` por medio de `Elemento`, y todos los elementos visuales en `Xamarin.Forms` definen la mayoría de sus propiedades como propiedades enlazables. Por esta razón, los objetivos de enlace de datos son casi siempre los elementos visuales o como se verá en el capítulo 19, "vistas" Colección -Objetos llamados *Células* que se traducen a elementos visuales.

A pesar de que el objetivo de un enlace de datos debe estar respaldada por una `BindableProperty` objeto, no hay tal requisito para una fuente de enlace de datos. La fuente puede ser una llanura C # propiedad de edad. Sin embargo, en todos menos en los enlaces de datos más triviales, un cambio en la propiedad de origen provoca un cambio correspondiente en la propiedad de destino. Esto significa que el objeto de origen debe implementar algún tipo de mecanismo de notificación para indicar cuando cambia la propiedad. Este mecanismo de notificación es el `INotifyProper-`

`tyChanged` interfaz, que es una interfaz estándar de .NET que participan en los enlaces de datos y se utiliza ampliamente para la implementación de la arquitectura MVVM.

La regla para una fuente, es decir, una fuente de enlace de datos de enlace de datos no trivial que puede cambiar dinámicamente, por lo tanto de valor es:

La fuente de un conjunto de datos no triviales de unión debe implementar `INotifyPropertyChanged`.

A pesar de su importancia, el `INotifyPropertyChanged` interfaz tiene la virtud de ser muy simple: consiste únicamente en un solo evento, denominado `PropertyChanged`, el cual se activa una clase cuando una propiedad ha cambiado.

Muy convenientemente para nuestros propósitos, `bindableObject` implementa `INotifyPropertyChanged`.

Cualquier propiedad que está respaldado por una propiedad enlazable se dispara automáticamente una `PropertyChanged` evento cuando que los cambios de propiedad. Este disparo automático del evento se extiende a las propiedades enlazables puede definir en sus propias clases.

Esto significa que se puede definir enlaces de datos entre las propiedades de los objetos visuales. En el gran

esquema de las cosas, la mayoría de los enlaces de datos probablemente vincular objetos visuales con los datos subyacentes, pero para fines de aprendizaje sobre los enlaces de datos y experimentando con ellos, es bueno para vincular simplemente propiedades de dos puntos de vista, sin definir las clases de datos.

Durante los primeros ejemplos de este capítulo, verá enlaces de datos en la que la fuente es el Valor **propiedad de una deslizador y el objetivo es la Opacidad propiedad de una Etiqueta. Como manipular el deslizador, el Etiqueta cambios de transparente a opaco. Ambas propiedades son de tipo doble y el rango de 0 a 1, por lo que son una combinación perfecta.**

Usted ya sabe cómo hacer este pequeño trabajo con un simple controlador de eventos. Vamos a ver cómo hacerlo con un enlace de datos.

Código y XAML

Aunque la mayoría de los enlaces de datos se definen en XAML, usted debe saber cómo hacer uno de cada código. He aquí una forma (pero no la única) para establecer la unión en un código de datos:

- Selecciona el **BindingContext** propiedad en el objeto de destino para referirse al objeto de origen.
- Llamada **SetBinding** en el objeto de destino para especificar tanto las propiedades de origen y destino.
los BindingContext propiedad se define por BindableObject. (Es el *sólo* Propiedad Definido por BindableObject). los SetBinding método también se define por bindableObject, pero hay dos sobrecargas adicionales de la SetBinding método en el BindableObjectExtensions clase. La propiedad de destino se especifica como una BindableProperty; la propiedad de origen a menudo se especifica como una cadena.

los OpacityBindingCode programa crea dos elementos, una Etiqueta y una deslizador, y define un enlace de datos que se dirige a la Opacidad propiedad de la Etiqueta desde el Valor propiedad de la deslizador.

```
público clase OpacityBindingCodePage : Pagina de contenido
{
    público OpacityBindingCodePage ()
    {
        Etiqueta etiqueta = nuevo Etiqueta
        {
            text = "Opacidad Encuadernación Demo",
            Tamaño de Letra = Dispositivo .GetNamedSize ( NamedSize .Grande, tipo de ( Etiqueta )),
            VerticalOptions = LayoutOptions .CenterAndExpand,
            HorizontalOptions = LayoutOptions .Centrar
        };

        deslizador deslizador = nuevo deslizador
        {
            VerticalOptions = LayoutOptions .CenterAndExpand
        };

        // Establecer el contexto de unión: diana es la etiqueta; fuente es Slider.
        label.BindingContext = cursor;
```

```
// enlazar las propiedades: objetivo de opacidad; es fuente de valor.  
label.SetBinding ( Etiqueta.OpacityProperty, "Valor" );  
  
// Construir la página.  
Relleno = nuevo Espesor (10, 0);  
content = nuevo StackLayout  
{  
    Los niños = (label, slider)  
};  
}  
}
```

Aquí está el valor de la propiedad que conecta los dos objetos:

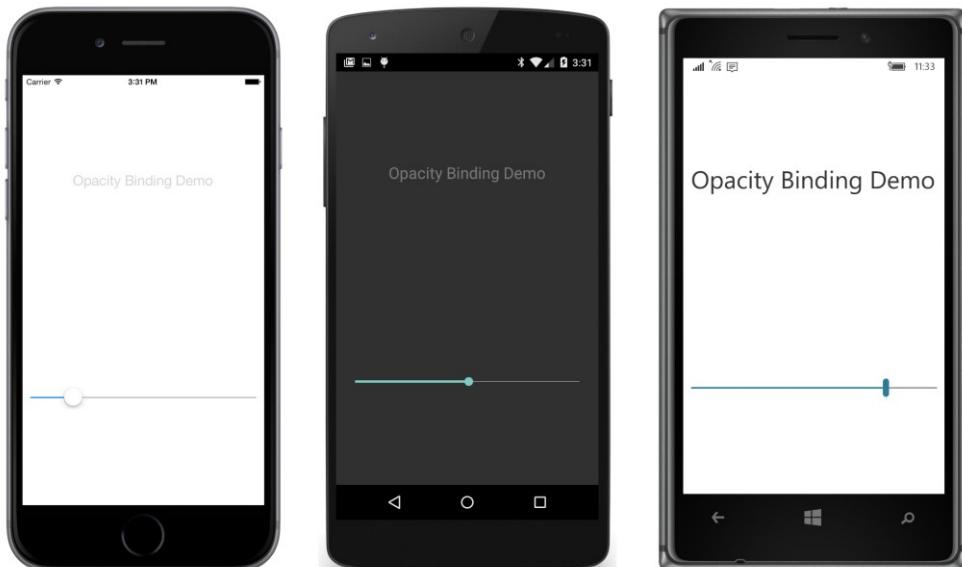
```
label.BindingContext = cursor;
```

los etiqueta objeto es el objetivo y el deslizador objeto es la fuente. Aquí está la llamada al método que une las dos propiedades:

```
label.SetBinding ( Etiqueta .OpacityProperty, "Valor" );
```

El primer argumento de `SetBinding` es de tipo `BindableProperty`, y ese es el requisito para la propiedad de destino. Sin embargo, la propiedad de origen no es más que especifica como una cadena. Puede ser cualquier tipo de propiedad.

La captura de pantalla demuestra que no es necesario establecer un controlador de eventos para utilizar la deslizadora para el control de otros elementos de la página:



Por supuesto, *alguien* es establecer un controlador de eventos. Bajo las sábanas, cuando la unión se inicializa, sino que también realiza la inicialización en el objetivo mediante el establecimiento de la Opacidad propiedad de la Etiqueta desde el

Valor propiedad de la Deslizador. (Como usted descubrió en el capítulo anterior, cuando se configura un controlador de eventos a sí mismo, esta inicialización no se produce automáticamente.) A continuación, el código comprueba si vinculantes internas del objeto de origen (en este caso el deslizador) implementa el `INotifyProperty`

cambiado interfaz. Si es así, una `PropertyChanged` gestora se establece en el Deslizador. Cada vez que el Valor cambios de propiedad, el enlace establece el nuevo valor a la Opacidad propiedad de la Etiqueta.

La reproducción de la unión en XAML implica dos extensiones de marcado que no ha visto todavía:

- `x:` Referencia, que es parte de la especificación XAML 2009.
- Unión, que es parte de las interfaces de usuario basadas en XAML de Microsoft.

los `x:` Referencia extensión de la unión es muy simple, pero el Unión extensión de marcado es el más extenso y complejo extensión de marcado en todas Xamarin.Forms. Se introdujo de forma incremental a lo largo de este capítulo.

He aquí cómo se establece el enlace de datos en XAML:

- **Selecciona el `BindingContext` propiedad del elemento de objetivo (la Etiqueta) a una `x:` Referencia extensión de marcado que hace referencia al elemento de origen (la Deslizador).**
- **Establecer la propiedad de destino (el Opacidad propiedad de la Etiqueta) a una Unión extensión de marcado que hace referencia a la propiedad de origen (el Valor propiedad de la Deslizador).**

los **OpacityBindingXaml** proyecto muestra el marcado completo:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns:x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " OpacityBindingXaml.OpacityBindingXamlPage "
    Relleno = " 10, 0 ">

< StackLayout >
    < Etiqueta Texto = " Opacidad de demostración Encuadernación "
        Tamaño de fuente = " Grande "
        VerticalOptions = " CenterAndExpand "
        HorizontalOptions = " Centrar "
        BindingContext = " {X: Name = Referencia deslizador} "
        Opacidad = " {Binding Path = Valor} " />

    < deslizador x: Nombre = " deslizador "
        VerticalOptions = " CenterAndExpand " />
</ StackLayout >
</ Página de contenido >
```

Las dos extensiones de marcado para la unión son los dos últimos valores de atributo en el Etiqueta. El archivo de código subyacente contiene nada más que la llamada norma de `InitializeComponent`.

Cuando se ajusta el `BindingContext` en el marcado, es muy fácil olvidar el `x:` Referencia extensión de marcado y simplemente especifica el nombre de la fuente, pero eso no funciona.

los Camino argumento de la Unión expresión de marcado especifica la propiedad de origen. ¿Por qué se llama este argumento Camino más bien que ¿Propiedad? Verá por qué más adelante en este capítulo.

Puede hacer que el marcado de un poco más corto. La clase pública que proporciona soporte para Referencia es ReferenceExtension, que define su propiedad de contenido para ser Nombre. La propiedad de contenido BindingExtension (que no es una clase pública) es Camino, por lo que no es necesario el Nombre y Camino argumentos y signos iguales:

```
<Etiqueta Texto = "Opacidad de demostración Encuadernación "
    Tamaño de fuente = "Grande "
    VerticalOptions = "CenterAndExpand "
    HorizontalOptions = "Centrar "
    BindingContext = "{X: deslizador de referencia} "
    Opacidad = "0 Valor Encuadernación ">
```

O si desea realizar el marcado ya, se puede romper el BindingContext y Opacidad propiedades como elementos de propiedad y les establecer utilizando la sintaxis elemento regular para x: Referencia y Unión:

```
<Etiqueta Texto = "Opacidad de demostración Encuadernación "
    Tamaño de fuente = "Grande "
    VerticalOptions = "CenterAndExpand "
    HorizontalOptions = "Centrar ">

    <Label.BindingContext>
        <x: Referencia Nombre = "deslizador ">/>
    </Label.BindingContext>

    <Label.Opacity>
        <Unión Camino = "Valor ">/>
    </Label.Opacity>
</Etiqueta >
```

Como verá, el uso de elementos de propiedad para los enlaces veces es conveniente en relación con el enlace de datos.

Fuente y BindingContext

los BindingContext propiedad es en realidad una de las dos formas de vincular los objetos de origen y de destino. Se puede prescindir, alternativamente, con BindingContext e incluir una referencia al objeto de origen dentro de la propia expresión de enlace.

los **BindingSourceCode** proyecto tiene una clase de página que es idéntica a la de **OpacityBindingCode** excepto que la unión se define en dos estados que no implican la BindingContext propiedad:

```
clase pública BindingSourceCodePage : Pagina de contenido
{
    público BindingSourceCodePage ()
    {
        Etiqueta etiqueta = nuevo Etiqueta
    }
}
```

```

text = "Opacidad Encuadernación Demo",
Tamaño de Letra = Dispositivo .GetNamedSize ( NamedSize .Grande, tipo de ( Etiqueta )),
VerticalOptions = LayoutOptions .CenterAndExpand,
HorizontalOptions = LayoutOptions .Centrar
};

deslizador deslizador = nuevo deslizador
{
    VerticalOptions = LayoutOptions .CenterAndExpand
};

// define objeto de enlace con el objeto de origen y la propiedad.
Unión = vinculante nuevo Unión
{
    Fuente = deslizador,
    path = "Valor"
};

// Enlazar la propiedad Opacidad de la etiqueta a la fuente.
label.SetBinding ( Etiqueta .OpacityProperty, la unión);

// Construir la página.
Relleno = nuevo Espesor (10, 0);
content = nuevo StackLayout
{
    Los niños = {label, slider}
};
}

}

```

El objeto de destino y propiedad todavía están especificados en la llamada a la SetBinding método:

```
label.SetBinding ( Etiqueta .OpacityProperty, la unión);
```

Sin embargo, el segundo argumento hace referencia a una Unión Objeto que especifica el objeto de origen y la propiedad:

```

Unión = vinculante nuevo Unión
{
    Fuente = deslizador,
    path = "Valor"
};

```

Esa no es la única manera de crear una instancia e inicializar una Unión objeto. Una extensa Unión constructor permite especificar muchos Unión propiedades. Así es como se podría utilizar en el BindingSourceCode programa:

```
Unión = vinculante nuevo Unión ( "Valor" , BindingMode .Defecto, nulo , nulo , nulo , Control deslizante);
```

O puede utilizar un argumento con nombre para hacer referencia al deslizador objeto:

```
Unión = vinculante nuevo Unión ( "Valor" , Fuente: control deslizante);
```

Unión También tiene un genérico Crear método que le permite especificar el Camino propiedad como una Func objeto en lugar de como una cadena de modo que es más inmune a los errores ortográficos o cambios en el nombre de la propiedad. Sin embargo, esta Crear método no incluye un argumento para la Fuente propiedad, por lo que es necesario establecer por separado:

```
Unión = vinculante Unión .Create < deslizador > (Src => src.Value);
binding.Source = cursor;
```

los BindableObjectExtensions clase define dos sobrecargas de SetBinding que le permiten evitar explícitamente instancias de un Unión objeto. Sin embargo, ninguna de estas sobrecargas incluye la Fuente propiedad, por lo que se limita a los casos en los que usted está utilizando el BindingContext.

los BindingSourceXaml programa demuestra cómo tanto el objeto de origen y la propiedad de origen pueden ser especificados en el Unión extensión de marcado:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " BindingSourceXaml.BindingSourceXamlPage "
    Relleno = " 10, 0 " >

    < StackLayout >
        < Etiqueta Texto = " Encuadernación Fuente Demostración "
            Tamaño de fuente = " Grande "
            VerticalOptions = " CenterAndExpand "
            HorizontalOptions = " Centrar "
            Opacidad = " {Binding Fuente = {x: Name = Referencia deslizador},
                Path = Valor} " />

        < deslizador x: Nombre = " deslizador "
            VerticalOptions = " CenterAndExpand " />
    </ StackLayout >
</ Página de contenido >
```

los Unión extensión de marcado ahora tiene dos argumentos, uno de los cuales es otra extensión de marcado para x: Referencia, por lo que un par de llaves se anidan dentro de las principales llaves:

```
Opacidad = " {Binding Fuente = {x: Name = Referencia deslizador},
    Path = Valor} " />
```

Para mayor claridad visual, los dos Unión argumentos están alineados verticalmente dentro de la extensión de marcado, pero eso no es necesario. Los argumentos deben estar separados por una coma (aquí al final de la primera línea), y sin comillas deben aparecer dentro de las llaves. Usted no está tratando con atributos XML dentro de la extensión de marcado. Estos son los argumentos extensión de marcado.

Se puede simplificar la extensión de marcado anidado al eliminar la Nombre nombre de argumento y signo de igualdad en x: Referencia porque Nombre es la propiedad de contenido de la ReferenceExtension clase:

```
Opacidad = " {Binding Fuente = {x: deslizador de referencia},
    Path = Valor} " />
```

Sin embargo, *no* poder eliminar de manera similar la Camino nombre de argumento y signo de igualdad. Aunque BindingExtension define Camino como su propiedad de contenido, el nombre del argumento sólo puede eliminarse

cuando ese argumento es el primero de los múltiples argumentos. Tienes que cambiar alrededor de los argumentos, así:

```
Opacidad = " {Binding Path = Valor,
Fuente = {x: deslizador de referencia}} " />
```

Y a continuación, se puede eliminar la Camino nombre de argumento, y tal vez pasar todo a una sola línea:

```
Opciedad = " {Binding Valor, Fuente = {x: deslizador de referencia}} " />
```

Sin embargo, debido a que el primer argumento es que falta un nombre de argumento y el segundo argumento tiene un nombre de argumento, toda la expresión se ve un poco rara, y podría ser difícil captar la Unión argumentos a primera vista. Además, tiene sentido para el Fuente a especificar *antes de* el Camino porque la propiedad particular especificada por el Camino sólo tiene sentido para un tipo particular de objeto, y que se especifica por el Fuente.

En este libro, siempre que el Unión extensión de marcado incluye una Fuente argumento, será en primer lugar, seguido por el Camino. De lo contrario, el Camino será el primer argumento, ya menudo el Camino nombre de argumento será eliminado.

Puede evitar el problema en su totalidad mediante la expresión Unión en forma de elemento:

```
< Etiqueta Texto = " Encuadernación Fuente Demostración "
  Tamaño de fuente = " Grande "
  VerticalOptions = " CenterAndExpand "
  HorizontalOptions = " Centrar " >
< Label.Opacity >
  < Unión Fuente = " {x: deslizador de referencia} "
    Camino = " Valor " />
</ Label.Opacity >
</ Etiqueta >
```

los x: Referencia todavía existe extensión de marcado, pero también se puede expresar en la forma que el elemento así:

```
< Etiqueta Texto = " Encuadernación Fuente Demostración "
  Tamaño de fuente = " Grande "
  VerticalOptions = " CenterAndExpand "
  HorizontalOptions = " Centrar " >
< Label.Opacity >
  < Unión Camino = " Valor " >
    < Binding.Source >
      < x: Referencia Nombre = " deslizador " />
    </ Binding.Source >
  </ Unión >
</ Label.Opacity >
</ Etiqueta >
```

Ahora ha visto dos formas de especificar el vínculo entre el objeto de origen con el objeto de destino:

- Utilizar el BindingContext para hacer referencia al objeto de origen.
- Utilizar el Fuente propiedad de la Unión clase o la Unión extensión de marcado.

Si especifica tanto, la Fuente la propiedad tiene prioridad sobre la BindingContext.

En los ejemplos que hemos visto hasta ahora, estas dos técnicas han sido más o menos intercambiables. Sin embargo, tienen algunas diferencias significativas. Por ejemplo, suponga que tiene un objeto con dos propiedades que son blancos de dos enlaces de datos diferentes que implican a dos objetos diferentes o fuente de ejemplo, una Etiqueta con el Opacidad propiedad unida a una deslizador y el Es visible propiedad unida a una Cambiar. No se puede utilizar BindingContext para ambas fijaciones PORQUE BindingContext se aplica a todo el objeto de destino y sólo puede especificar una sola fuente. Debe utilizar el Fuente propiedad de

Unión por lo menos uno de estos enlaces.

BindingContext está a su vez respaldado por una propiedad enlazable. Esto significa que BindingContext se puede fijar de una Unión extensión de marcado. Por el contrario, no se puede establecer la Fuente propiedad de Unión a otro Unión porque Unión no se deriva de bindableObject, lo que significa Fuente no está respaldado por una propiedad enlazable y por lo tanto no puede ser el objetivo de un enlace de datos.

En esta variación de la BindingSourceXaml marcado, el BindingContext propiedad de la Etiqueta se establece en un Unión extensión de marcado que incluye una Fuente y Camino.

```
<Etiqueta Texto = "Encuadernación Fuente Demostración"
    Tamaño de fuente = "Grande"
    VerticalOptions = "CenterAndExpand"
    HorizontalOptions = "Centrar"
    BindingContext = "{Binding Fuente = {x: Name = Referencia deslizador},
        Path = Valor}"
    Opacidad = "{Unión}" />
```

Esto significa que el BindingContext para este Etiqueta No es el deslizador objeto como en los ejemplos anteriores, pero la doble eso es el Valor propiedad de la Deslizador. Para obligar a la Opacidad pertenencia a esta duplicó con bie, todo lo que se requiere es una lata vacía Unión extensión de marcado que básicamente dice "utilizar el EnlazamientoContext para toda la fuente de enlace de datos".

Quizás la diferencia más importante entre BindingContext y Fuente es una característica muy especial que hace BindingContext a diferencia de cualquier otra propiedad en todas Xamarin.Forms:

El contexto de unión se propaga a través del árbol visual.

En otras palabras, si se establece BindingContext en un StackLayout, se aplica a todos los niños de esa StackLayout y sus hijos también. Los enlaces de datos dentro de ese StackLayout no tiene que especificar BindingContext o el Fuente argumento para Unión. heredan BindingContext desde el StackLayout. O los hijos de la StackLayout puede anular que heredaron BindingContext

con BindingContext la configuración de su cuenta o con una Fuente establecer en sus fijaciones.

Esta característica resulta ser excepcionalmente útil. Supongamos que una StackLayout contiene un montón de efectos visuales con enlaces de datos establecidos para varias propiedades de una clase particular. Selecciona el BindingContext propiedad de ese StackLayout. A continuación, los enlaces de datos individuales sobre los niños de la StackLayout no requieren ya sea una Fuente especificación o una BindingContext ajuste. A continuación, puede establecer el EnlazamientoContext del StackLayout a diferentes instancias de esa clase para mostrar las propiedades de cada

ejemplo. Vas a ver ejemplos de esta técnica y otras maravillas de enlace de datos en los capítulos por delante, y en particular en el capítulo 19.

Mientras tanto, vamos a ver un ejemplo mucho más simple de BindingContext propagación a través del árbol visual.

Los WebView tiene la intención de incorporar un navegador web dentro de su aplicación. Como alternativa, puede utilizar WebView en conjunción con el HtmlWebViewSource clase para mostrar un trozo de HTML, tal vez guardado como un recurso incrustado en el PCL.

Para la visualización de páginas web, se utiliza WebView con el UriWebViewSource clase para especificar una dirección URL inicial. Sin embargo, UriWebViewSource y HtmlWebViewSource ambos derivan de la clase abstracta

WebViewSource, y que clase define una conversión implícita de cuerda y Uri a sí mismo, así que todo lo que realmente necesita hacer es crear una cadena con una dirección web a la Fuente propiedad de WebView dirigir Web-

Ver para presentar esa página web.

WebView también define dos métodos, denominados Regresa y GoForward, que internamente implementar el Espalda y Adelante botones se encuentran típicamente en los navegadores web. Su programa necesita saber cuando se puede permitir que estos botones, por lo WebView también define dos sólo para obtener las propiedades booleanas, nombrados CanGoBack y CanGoForward. Estas dos propiedades están respaldados por propiedades enlazables, lo que significa que cualquier cambio en estas propiedades dan lugar a PropertyChanged eventos que se dispararon, lo que significa, además, que puedan ser utilizados como fuentes para activar y desactivar los botones de enlace de datos.

Aquí está el archivo XAML para WebViewDemo. Observe que el anidado StackLayout que contiene los dos Botón elementos tiene su BindingContext propiedad establecida en el WebView. Los dos Botón niños de ese StackLayout heredar el BindingContext, por lo que los botones pueden tener muy sencillo Unión expresiones en sus Están habilitado propiedades que hacen referencia a sólo el CanGoBack y CanGoForward

propiedades:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " WebViewDemo.WebViewDemoPage " >

< ContentPage.Padding >
    < OnPlatform x: TypeArguments = " Espesor "
        iOS = " 10, 20, 10, 0 "
        Androide = " 10, 0 "
        WinPhone = " 10, 0 " />
</ ContentPage.Padding >

< StackLayout >
    < Entrada Teclado = " url "
        marcador de posición = " dirección web "
        Terminado = " OnEntryCompleted " />

    < StackLayout Orientación = " Horizontal "
        BindingContext = " {X: Referencia web View} " >

        < Botón Texto = " & # X21D0; "
            Tamaño de fuente = " Grande " >
```

```

HorizontalOptions = "FillAndExpand"
Está habilitado = "true" La unión CanGoBack
hecho clic = "OnGoBackClicked" />

< Botón Texto = " & # X21D2;" 
    Tamaño de fuente = "Grande"
    HorizontalOptions = "FillAndExpand"
    Está habilitado = "true" La unión CanGoForward
    hecho clic = "OnGoForwardClicked" />

</ StackLayout >

< WebView x: Nombre = "webView" 
    VerticalOptions = "FillAndExpand"
    Fuente = "https://xamarin.com" />

</ StackLayout >
</ Página de contenido >

```

El archivo de código subyacente necesita para manejar la hecho clic eventos para la **Espalda** y **Adelante** botones, así como el Terminado para el caso Entrada que le permite introducir una dirección web de su propia:

```

pública clase parcial WebViewDemoPage : Página de contenido
{
    pública WebViewDemoPage ()
    {
        InitializeComponent ();
    }

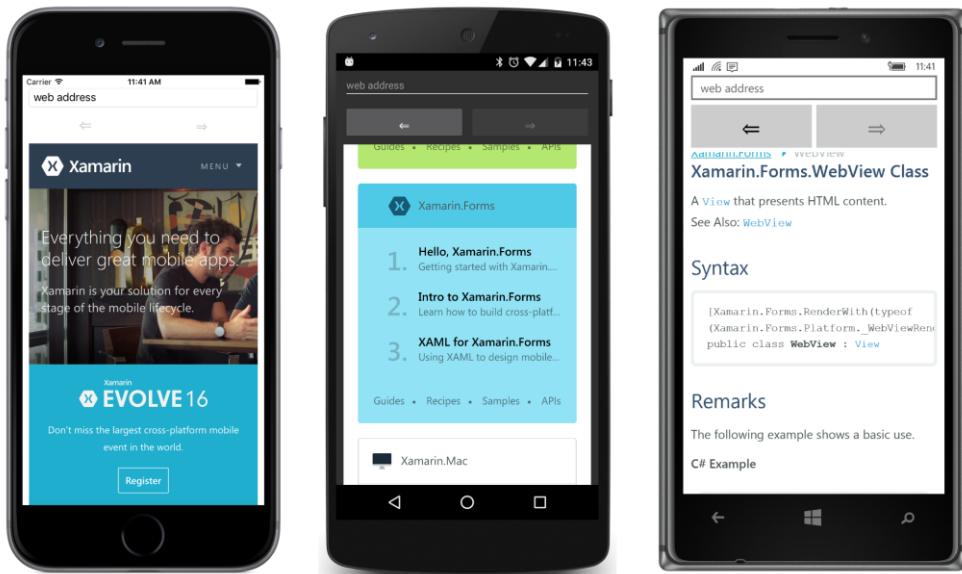
    vacío OnEntryCompleted ( objeto remitente, EventArgs args)
    {
        webView.Source = (( Entrada ) Remitente) .Texto;
    }

    vacío OnGoBackClicked ( objeto remitente, EventArgs args)
    {
        webView.GoBack ();
    }

    vacío OnGoForwardClicked ( objeto remitente, EventArgs args)
    {
        webView.GoForward ();
    }
}

```

No es necesario introducir una dirección web cuando el programa se pone en marcha porque el archivo XAML está codificado para ir a su sitio web favorito, y se puede navegar alrededor de allí:



El modo de unión

Aquí es una Etiqueta cuyo Tamaño de fuente la propiedad está ligado a la Valor propiedad de una deslizador:

```
<Etiqueta Tamaño de fuente = "{Binding Fuente = {x: deslizador de referencia},
                                         Path = Valor}" />
<deslizador x: Nombre = "deslizador"
             Máximo = "100" />
```

Eso debería funcionar, y si lo pruebas, que va a funcionar. Usted será capaz de cambiar la Tamaño de fuente del Etiqueta mediante la manipulación de la Deslizador.

Pero aquí hay un Etiqueta y deslizador con la unión invertidos. En vez de Tamaño de fuente propiedad de la Etiqueta siendo el objetivo, ahora Tamaño de fuente es la fuente de los datos de unión, y el objetivo es la Valor propiedad de la deslizador:

```
<Etiqueta x: Nombre = "etiqueta" />
<deslizador Máximo = "100"
             Valor = "{Binding Fuente = {x: etiqueta Referencia},
                                         Path = Tamaño de Letra}" />
```

Eso no parece tener ningún sentido. Pero si lo intentas, que funcionará muy bien. Una vez más, la deslizador manipulará la Tamaño de fuente propiedad de la Etiqueta.

Las obras de la segunda unión a causa de algo llamado el *modo de unión*.

Usted ha aprendido que un enlace de datos establece el valor de una propiedad de destino a partir del valor de una fuente

propiedad, pero a veces el flujo de datos no es tan clara. La relación entre el objetivo y la fuente está definido por los miembros de la **BindingMode** enumeración:

- Defecto
- Un camino - cambios en la fuente afectan a la diana (normal).
- OneWayToSource - cambios en el objetivo afectan a la fuente.
- TwoWay - cambios en el origen y el destino se afectan entre sí.

Esta **BindingMode** enumeración juega un papel en dos clases diferentes:

Cuando se crea una BindableProperty objeto mediante el uso de uno de los estático Crear o CreateRead-

Solamente métodos estáticos, puede especificar un valor predeterminado **BindingMode** valor que debe utilizarse cuando la propiedad es el objetivo de un enlace de datos.

Si no se especifica nada, el modo por defecto es vinculante Un camino para las propiedades enlazables que son de lectura y escritura, y OneWayToSource para las propiedades enlazables de sólo lectura. Si especifica **Enlazar-ingMode.Default** al crear una propiedad enlazable, el modo de enlace predeterminada para la propiedad se establece en Un camino. (En otras palabras, el **BindingMode.Default** miembro no está pensado para definir propiedades enlazables.)

Puede anular que modo de unión por defecto para la propiedad de destino cuando se define una unión ya sea en clave o XAML.

Anula el modo de enlace de selección, definiendo la **Modo** propiedad de **Enlazar-**

En g a uno de los miembros de la **BindingMode** enumeración. los **Defecto** miembro significa que desea utilizar el modo de enlace predeterminada definida para la propiedad de destino.

Cuando se establece el Modo propiedad a OneWayToSource usted está *no* commutación de la diana y la fuente. El objetivo sigue siendo el objeto sobre el que se ha fijado la **BindingContext y la propiedad en la que ha llamado **SetBinding** o aplicado el **Unión extensión** de marcado. Pero los flujos de datos en una dirección-de la meta a la fuente diferente.**

La mayoría de las propiedades enlazables tienen un modo de unión por defecto Un camino. Sin embargo hay algunas excepciones. De los puntos de vista que le ha surgido hasta ahora en este libro, las siguientes propiedades tienen un modo por defecto TwoWay:

Clase	La propiedad que es TwoWay
deslizador	Valor
paso a paso	Valor
Cambiar	IsToggled
Entrada	Texto
Editor	Texto
Barra de búsqueda	Texto
Selector de fechas	Fecha
TimePicker	Hora

Las propiedades que tienen un modo de unión por defecto TwoWay son los más propensos a ser utilizado con los modelos de datos subyacentes en un escenario MVVM. Con MVVM, los objetivos vinculantes son objetos visuales y

las fuentes de unión son objetos de datos. En general, desea que los datos fluyan en ambos sentidos. Desea que los objetos visuales para mostrar los valores de los datos subyacentes (del origen al destino), y desea que los objetos visuales interactivos para causar cambios en los datos subyacentes (objetivo a la fuente).

los BindingModes programa se conecta cuatro Etiqueta elementos y cuatro deslizador elementos con consolidaciones "normales", es decir que el objetivo es la Tamaño de fuente propiedad de la Etiqueta y la fuente es la Valor propiedad de la deslizador:

```
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns:x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " BindingModes.BindingModesPage "
    Relleno = " 10, 0 " >
```

```
< ContentPage.Resources >
    < ResourceDictionary >
        < Estilo Tipo de objetivo = " StackLayout " >
            < Setter Propiedad = " VerticalOptions " Valor = " CenterAndExpand " />
        </ Estilo >

        < Estilo Tipo de objetivo = " Etiqueta " >
            < Setter Propiedad = " HorizontalOptions " Valor = " Centrar " />
        </ Estilo >
    </ ResourceDictionary >
</ ContentPage.Resources >

< StackLayout VerticalOptions = " Llenar " >
    < StackLayout >
        < Etiqueta Texto = " Defecto " 
            Tamaño de fuente = " {Binding Fuente = {x: slider1 Referencia},
                Path = Valor} " />
        < deslizador x: Nombre = " slider1 " 
            Máximo = " 50 " />
    </ StackLayout >

    < StackLayout >
        < Etiqueta Texto = " Un camino " 
            Tamaño de fuente = " {Binding Fuente = {x: slider2 Referencia},
                Path = Valor,
                Mode = OneWay} " />
        < deslizador x: Nombre = " slider2 " 
            Máximo = " 50 " />
    </ StackLayout >

    < StackLayout >
        < Etiqueta Texto = " OneWayToSource " 
            Tamaño de fuente = " {Binding Fuente = {x: slider3 Referencia},
                Path = Valor,
                Mode = OneWayToSource} " />
        < deslizador x: Nombre = " slider3 " 
            Máximo = " 50 " />
    </ StackLayout >

    < StackLayout >
```

```
< Etiqueta Texto = "TwoWay"
  Tamaño de fuente = "{Binding Fuente = {x: slider4 Referencia},
  Path = Valor,
  Mode = TwoWay} " />
</ deslizador x: Nombre = "slider4"
  Máximo = " 50 " />
</ StackLayout >
</ StackLayout >
</ Página de contenido >
```

los Texto del Etiqueta indica el modo de unión. La primera vez que ejecute este programa, toda la deslizador elementos se inicializan a cero, excepto para la tercera, que es ligeramente diferente de cero:



Mediante la manipulación de cada uno deslizador, se puede cambiar el Tamaño de fuente del Etiqueta, pero no funciona para la tercera porque el OneWayToSource de modo indica que los cambios en la Diana (la Tamaño de fuente propiedad de la Etiqueta) afectar a la fuente (la Valor propiedad de la Deslizador):



Aunque no es muy evidente en este caso, el modo por defecto es vinculante. Un camino porque la unión se establece en el Tamaño de fuente propiedad de la Etiqueta, y ese es el modo de encuadernación por el impago Tamaño de fuente propiedad.

los **ReverseBinding** programa establece los enlaces en la Valor propiedad de la deslizador:

```
< Página de contenido xmlns = "http://xamarin.com/schemas/2014/forms"
    xmlns:x = "http://schemas.microsoft.com/winfx/2009/xaml"
    x: Class = "ReverseBinding.ReverseBindingPage"
    Relleno = "10, 0" >

< ContentPage.Resources >
    < ResourceDictionary >
        < Estilo Tipo de objetivo = "StackLayout" >
            < Setter Propiedad = "VerticalOptions" Valor = "CenterAndExpand" />
        </ Estilo >

        < Estilo Tipo de objetivo = "Etiqueta" >
            < Setter Propiedad = "HorizontalOptions" Valor = "Centrar" />
        </ Estilo >
    </ ResourceDictionary >
</ ContentPage.Resources >

< StackLayout VerticalOptions = "Llenar" >
    < StackLayout >
        < Etiqueta x: Nombre = "label1"
            Texto = "Defecto" />
        < deslizador Máximo = "50"
            Valor = "{Binding Fuente = {x: label1 Referencia},
            Path = Tamaño de Letra}" />
    </ StackLayout >
```

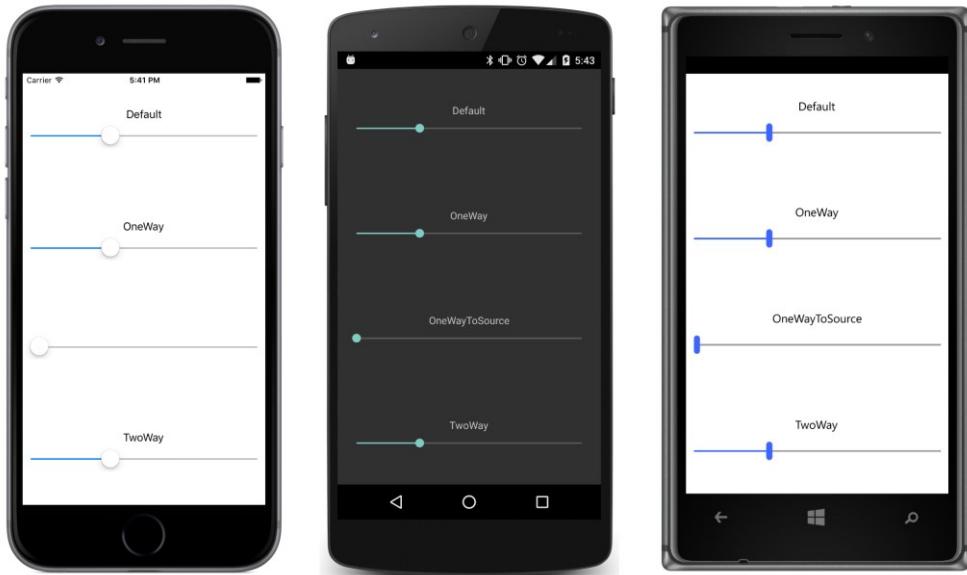
```
< StackLayout >
    < Etiqueta x: Nombre = "label2" 
        Texto = "Un camino" />
    < deslizador Máximo = "50" 
        Valor = "{Binding Fuente = {x: label2 Referencia},
                    Path = Tamaño de Letra,
                    Mode = OneWay}" />
</ StackLayout >

< StackLayout >
    < Etiqueta x: Nombre = "LABEL3" 
        Texto = "OneWayToSource" />
    < deslizador Máximo = "50" 
        Valor = "{Binding Fuente = {x: LABEL3 Referencia},
                    Path = Tamaño de Letra,
                    Mode = OneWayToSource}" />
</ StackLayout >

< StackLayout >
    < Etiqueta x: Nombre = "Label4" 
        Texto = "TwoWay" />
    < deslizador Máximo = "50" 
        Valor = "{Binding Fuente = {x: Label4 Referencia},
                    Path = Tamaño de Letra,
                    Mode = TwoWay}" />
</ StackLayout >
</ StackLayout >
</ Página de contenido >
```

El modo de enlace predeterminada en estos enlaces es TwoWay porque ese es el modo establecido en el `BindableProperty.Create` método para la `Valor` propiedad de la Deslizador.

Lo interesante de este enfoque es que para tres de los casos aquí, el `Valor` propiedad de la deslizador se inicializa de la `Tamaño de fuente` propiedad de la Etiqueta:



Esto no sucede de OneWayToSource porque para ese modo, los cambios en la Valor propiedad de la deslizador afectar el Tamaño de fuente propiedad de la Etiqueta pero no a la inversa.

Ahora vamos a empezar la manipulación de las barras de desplazamiento:



Ahora el OneWayToSource trabaja porque los cambios en la unión Valor propiedad de la deslizador

afectar el Tamaño de fuente propiedad de la Etiqueta, pero el Un camino la unión no funciona debido a que indica que el Valor propiedad de la deslizador sólo se ve afectado por los cambios en la Tamaño de fuente propiedad de la Etiqueta.

El que la unión funciona mejor? El que la unión inicializa el Valor propiedad de la deslizador al Tamaño de fuente propiedad de la Etiqueta, sino que también permite deslizador manipulaciones para cambiar el Tamaño de fuente? Es la inversa en el conjunto de asignaciones deslizador con un modo de TwoWay, que es el modo por defecto.

Este es exactamente el tipo de inicialización deseada ver cuando una deslizador está obligado a algunos datos. Por esa razón, cuando se utiliza una deslizador con MVVM, la unión se establece en el deslizador a la vez que aparezca el valor de los datos y para manipular el valor de los datos.

El formato de cadenas

Algunos de los programas de ejemplo en los capítulos anteriores se utiliza controladores de eventos para mostrar los valores actuales de la deslizador y paso a paso puntos de vista. Si intenta definir un enlace que apunta a los datos Texto propiedad de una Etiqueta desde el Valor propiedad de una deslizador, Usted descubrirá que funciona, pero no tiene mucho control sobre él. En general, usted querrá controlar cualquier conversión de conversión de tipo o valor requerido en enlaces de datos. Eso se explica más adelante en este capítulo.

El formato de cadenas es especial, sin embargo. los Unión clase tiene una StringFormat propiedad que le permite incluir toda una cadena de formato .NET. Casi siempre, el objetivo de una tal unión es la Texto propiedad de una Etiqueta, pero el origen de enlace puede ser de cualquier tipo.

La cadena de formato de .NET que se proporciona a StringFormat deben ser adecuados para una llamada a la String.Format método estático, lo que significa que debe contener un marcador de posición de "{0}" con o sin una especificación de formato adecuado para el tipo, por ejemplo de datos de origen "{0: F3}" para mostrar una duplicó conble con tres cifras decimales.

En XAML, este marcador de posición es un poco de un problema, ya que las llaves pueden confundirse con las llaves utilizadas para delimitar extensiones de marcado. La solución más fácil es poner toda la cadena de formato entre comillas simples.

los ShowViewValues programa contiene cuatro ejemplos que muestran los valores actuales de una deslizador, Entrada, paso a paso, y Cambiar. Los códigos hexadecimales en la cadena de formato que se utilizan para la visualización de la Entrada contenidos son identificadores de Unicode para "comillas inteligentes":

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " ShowViewValues.ShowViewValuesPage "
    Relleno = " 10, 0 ">

    < StackLayout >
        < StackLayout VerticalOptions = " CenterAndExpand " >
            < Etiqueta Texto = " {Binding Fuente = {x: deslizador de referencia},
                Path = Valor, }
```

```
StringFormat = 'El valor Slider es {0: F3})' />
</deslizador x: Nombre = "deslizador" />
</StackLayout>

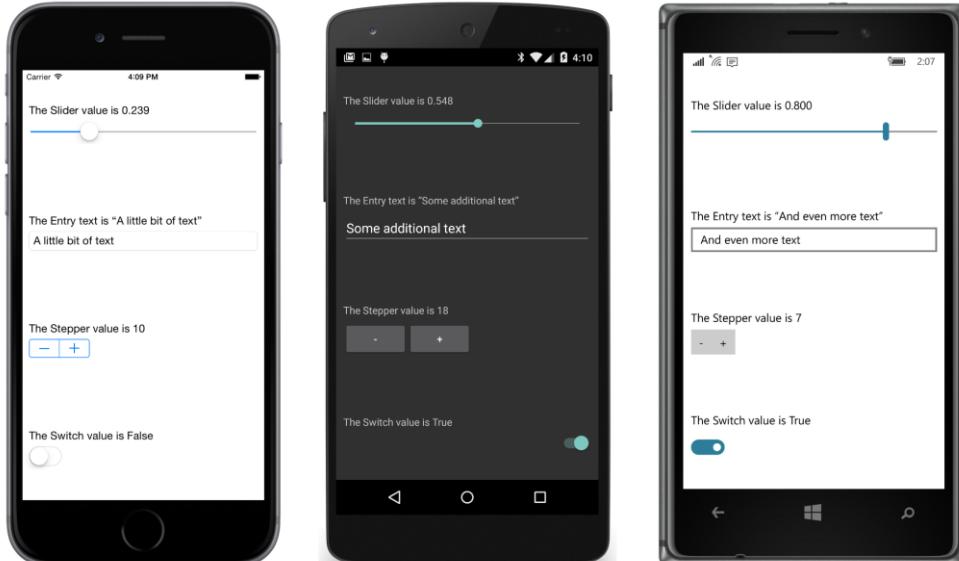
<StackLayout VerticalOptions = "CenterAndExpand" >
<Etiqueta Texto = "{Binding Fuente = {x: entrada Referencia},
Path = texto,
StringFormat = 'El texto de entrada es &# X201C; {0} &# X201D; }" />
<Entrada x: Nombre = "entrada" />
</StackLayout>

<StackLayout VerticalOptions = "CenterAndExpand" >
<Etiqueta Texto = "{Binding Fuente = {x: paso a paso Referencia},
Path = Valor,
StringFormat = 'El valor paso a paso es {0})" />
<paso a paso x: Nombre = "paso a peso" />
</StackLayout>

<StackLayout VerticalOptions = "CenterAndExpand" >
<Etiqueta Texto = "{Binding Fuente = {x: Interruptor de referencia},
Path = IsToggled,
StringFormat = 'El valor del interruptor es {0})" />
<Cambiar x: Nombre = "cambiar" />
</StackLayout>
</StackLayout>
</Página de contenido>
```

Cuando usas StringFormat es necesario prestar especial atención a la colocación de las comas, comillas simples, y llaves.

Aquí está el resultado:



Se puede recordar la **Qué tamaño** programa desde el capítulo 5, "Tratar con los tamaños". Este programa utiliza una **SizeChanged** controlador de eventos en la página para mostrar la anchura actual y la altura de la pantalla con unidades independientes del dispositivo.

los **WhatSizeBindings** programa hace todo el trabajo en XAML. En primer lugar se añade una **x: Nombre** atribuir a la etiqueta raíz para dar la **WhatSizeBindingsPage** objeto un nombre de página. Tres Etiqueta vistas comparten una horizontal **StackLayout** en el centro de la página, y dos de ellos tienen enlaces con el Anchura y

Altura propiedades. Los Anchura y Altura propiedades se consiguen solamente, sino que están respaldados por propiedades que pueden vincularse, por lo que el fuego **PropertyChanged** eventos cuando cambian:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " WhatSizeBindings.WhatSizeBindingsPage "
    x: Nombre = " página " >

    < StackLayout Orientación = " Horizontal "
        Espaciado = " 0 "
        HorizontalOptions = " Centrar "
        VerticalOptions = " Centrar " >

        < StackLayout.Resources >
            < ResourceDictionary >
                < Estilo Tipo de objetivo = " Etiqueta " >
                    < Setter Propiedad = " Tamaño de fuente " Valor = " Grande " />
                </ Estilo >
            </ ResourceDictionary >
        </ StackLayout.Resources >

        < Etiqueta Texto = " {Binding Fuente = {x: Página de referencia},
            Path = Ancho,
            StringFormat = '{0: F0}'} " />

        < ! - Signo de multiplicación. ->
        < Etiqueta Texto = " & # X00D7; " />

        < Etiqueta Texto = " {Binding Fuente = {x: Página de referencia},
            Path = Altura,
            StringFormat = '{0: F0}'} " />
    </ StackLayout >
</ Página de contenido >
```

Aquí está el resultado de los dispositivos utilizados para este libro:



La pantalla cambia a medida que gira el teléfono entre los modos vertical y horizontal.

Alternativamente, el `BindingContext` sobre el `StackLayout` se podría establecer a una `x: Referencia extensión de marcado` referencia a la página objeto, y el Fuente configuración de los enlaces no serían necesarios.

Por qué se llama “Camino”?

los Unión clase define una propiedad denominada Camino que se utiliza para establecer el nombre de propiedad de origen. Pero por qué se llama ¿Camino? ¿Por qué no se llama ¿Propiedad?

los Camino propiedad se llama como se llama, ya que no tiene por qué ser una propiedad. Puede ser una pila de propiedades, subpropiedades, y indizadores incluso conectados con períodos.

Utilizando Camino de esta manera puede ser complicado, así que aquí tiene un programa llamado `BindingPathDemos` que tiene cuatro Unión extensiones marcado, cada una de las cuales establece la Camino argumento para una cadena de nombres de propiedades e indexadores:

```
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    xmlns: Globo = " clr-espacio de nombres: System.Globalization; montaje = mscorelib "
    x: Class = " BindingPathDemos.BindingPathDemosPage "
    x: Nombre = " página " >

< ContentPage.Padding >
    < OnPlatform x: TypeArguments = " Espesor "
        iOS = " 10, 20, 10, 0 "
        Androide = " 10, 0 "
        WinPhone = " 10, 0 " />
```

```

</ ContentPage.Padding >

< ContentPage.Resources >
    < ResourceDictionary >
        < Estilo x: Key = " baseStyle " Tipo de objetivo = " Ver " >
            < Setter Propiedad = " VerticalOptions " Valor = " CenterAndExpand " />
        </ Estilo >

        < Estilo Tipo de objetivo = " Etiqueta " Residencia en = " {} StaticResource baseStyle " >
            < Setter Propiedad = " Tamaño de fuente " Valor = " Grande " />
            < Setter Propiedad = " HorizontalTextAlignment " Valor = " Centrar " />
        </ Estilo >

        < Estilo Tipo de objetivo = " deslizador " Residencia en = " {} StaticResource baseStyle " />
    </ ResourceDictionary >
</ ContentPage.Resources >

< StackLayout BindingContext = " {X: Página de referencia} " >
    < Etiqueta Texto = " {Binding Path = Padding.Top,
        StringFormat = 'El relleno superior es {0}'} " />

    < Etiqueta Texto = " {Binding Path = Content.Children [4].Value,
        StringFormat = 'El valor del deslizador es {0:F2}'} " />

    < Etiqueta Texto = " {Binding Fuente = {x: Globo estático: CultureInfo.CurrentCulture},
        Path = DateTimeFormat.DayNames [3],
        StringFormat = 'El día medio de la semana es {0}') " />

    < Etiqueta Texto = " {Binding Path = Content.Children [2].Text.Length,
        StringFormat = 'El Label anterior tiene {0} caracteres'} " />
    < deslizador />
</ StackLayout >
</ Página de contenido >

```

Sólo un elemento de aquí tiene una x: Nombre, y eso es la propia página. los BindingContext del StackLayout es que la página, por lo que todos los enlaces dentro de la StackLayout son en relación con la página (a excepción de la unión que tiene una explícita Fuente conjunto de propiedades).

El primero Unión Se ve como esto:

```

< Etiqueta Texto = " {Binding Path = Padding.Top,
        StringFormat = 'El relleno superior es {0}'} " />

```

los Camino comienza con la Relleno propiedad de la página. Esa propiedad es de tipo Espesor, por lo que es posible acceder a una propiedad de la Espesor estructura con un nombre de propiedad, tales como Parte superior. Por supuesto, Espesor es una estructura y por lo tanto no se deriva de bindableObject, así que Parte superior no puede ser una BindableProperty. La infraestructura de la unión no se puede establecer una PropertyChanged handler en esa propiedad, pero llevará a una PropertyChanged manejador en el Relleno propiedad de la página, y si eso cambia, la unión se actualizará el objetivo.

El segundo Unión hace referencia a la Contenido propiedad de la página, que es el StackLayout.

Ese StackLayout tiene un Niños propiedad, que es una colección, por lo que puede ser indexado:

```
<Etiqueta Texto = " {Binding Path = Content.Children [4] .Value,
StringFormat = 'El valor del deslizador es {0: F2} } " />
```

La vista en el índice 4 de la Niños es una colección deslizador (hacia abajo en la parte inferior del margen de beneficio, sin atributos establecidos), que tiene una Valor propiedad, y eso es lo que se muestra aquí.

El tercero Unión anula su heredado BindingContext mediante el establecimiento de la Fuente argumento a una propiedad estática por medio de x: Estático. los globo prefijo se define en la etiqueta raíz para referirse a la .NET Sys

tem.Globalization espacio de nombres, y de la Fuente se establece en el CultureInfo objeto que encapsula la cultura de teléfono del usuario:

```
<Etiqueta Texto = " {Binding Fuente = {x: Globo estático: CultureInfo.CurrentCulture},
Path = DateTimeFormat.DayNames [3],
StringFormat = 'El día medio de la semana es {0} } " />
```

Una de las propiedades de CultureInfo es DateTimeFormat, el cual es un DateTimeFormatInfo objeto que contiene información sobre la fecha y el formato del tiempo, incluyendo una propiedad denominada dayNames es una matriz de los siete días de la semana. El índice 3 escoge el del medio.

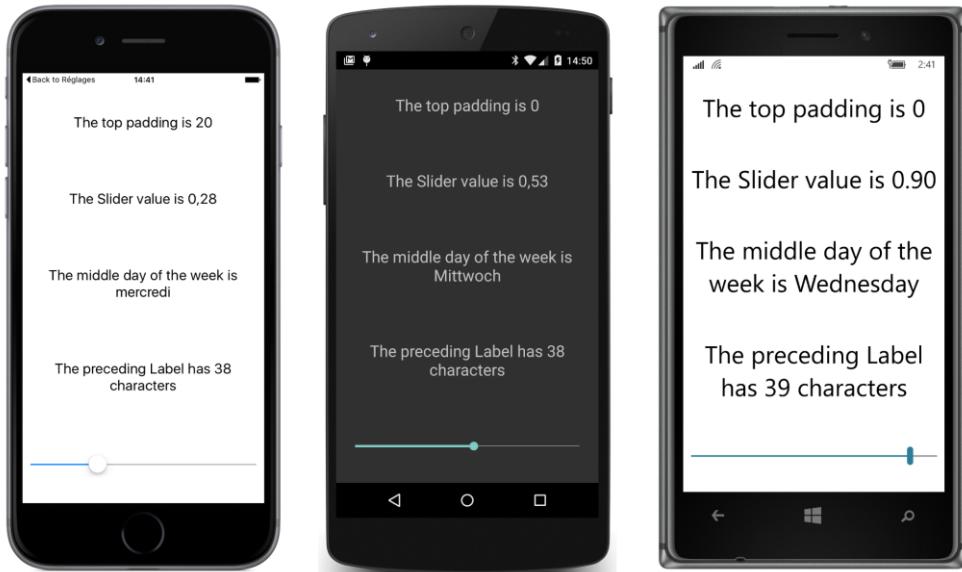
Ninguna de las clases en el System.Globalization espacio de nombres implementar INotifyProperty- cambiado, pero eso está bien porque los valores de estas propiedades no cambian en tiempo de ejecución.

El final Unión hace referencia al hijo de la StackLayout con un índice menor de 2. Esa es la anterior Etiqueta. Tiene un Texto propiedad, que es de tipo cuerda, y cuerda tiene un Longitud propiedad:

```
<Etiqueta Texto = " {Binding Path = Content.Children [2] .Text.Length,
StringFormat = 'El Label anterior tiene {0} caracteres'} " />
```

El sistema de unión se instala un controlador de propiedad cambiado para Texto propiedad de la Etiqueta, por lo que si cambia, la unión conseguirá la nueva longitud.

Para las siguientes capturas de pantalla, el teléfono IOS se cambió al francés, y el teléfono Android se cambió al alemán. Esto afecta el formato de la deslizador valor notar la coma en lugar de un período para el divisor decimal y el nombre del día medio de la semana:



Estas Camino especificaciones pueden ser difíciles de configurar y depurar. Tenga en cuenta que los nombres de clases no aparecen en el Camino especificaciones sólo para nombres de propiedades e indexadores. También hay que tener en cuenta que se puede construir una Camino especificación incremental, las pruebas de cada nueva pieza con un marcador de posición de "{0}" en StringFormat. Esto a menudo mostrar el nombre completo de la clase del tipo del valor establecido en la última propiedad en el Camino especificación, y que puede ser información muy útil.

Usted también querrá mantener un ojo en el Salida ventana en Visual Studio o Xamarin Studio cuando se ejecuta el programa en el depurador. Verá allí mensajes relacionados con la gestión del tiempo los errores encontrados por la infraestructura de la unión.

Encuadernación convertidores de valores

Ahora ya sabe cómo convertir cualquier objeto de origen de unión a una cadena utilizando StringFormat. Pero ¿qué pasa con otras conversiones de datos? Tal vez usted está utilizando una deslizador para un origen de enlace pero el objetivo está a la espera un número entero en lugar de una doble. O tal vez desea mostrar el valor de una Cambiar como texto, pero desea "Sí" y "No" en lugar de "verdadero" y "falso".

La herramienta para este trabajo es una clase de una clase a menudo informalmente muy pequeña llama *convertidor de valores* o (a veces) una *Conversor de unión*. Más formalmente, tal clase implementa el IValueConverter interfaz. Esta interfaz se define en el Xamarin.Forms espacio de nombres, pero es similar a una interfaz disponible en entornos basados en XAML de Microsoft.

Un ejemplo: A veces las aplicaciones necesitan para activar o desactivar una Botón basado en la presencia de texto en una Entrada. Quizás el Botón se etiqueta **Salvar** y el Entrada es un nombre de archivo. O el Botón es

etiquetado Enviar y el Entrada contiene un destinatario de correo. Los Botón no debe ser permitido a menos que el Entrada contiene al menos un carácter de texto.

Hay un par de maneras de hacer este trabajo. En un capítulo posterior, verá cómo un activador de datos puede hacerlo (y también puede realizar comprobaciones de validez del texto en el Entrada). Sin embargo, para este capítulo, vamos a hacerlo con un convertidor de valores.

El objetivo de enlace de datos es la Está habilitado propiedad de la Botón. Esta propiedad es de tipo bool. La fuente de unión es la Texto propiedad de una Entrada, o más bien la Longitud propiedad de ese Texto propiedad. Ese Longitud propiedad es de tipo Ent. El convertidor de valores necesita para convertir una Ent igual a 0 a una bool de falso y una positiva Ent a una bool de cierto. El código es trivial. Sólo tenemos que envolver en una clase que implementa IValueConverter.

Aquí es que la clase en el Xamarin.FormsBook.Toolkit biblioteca, con utilizando directivas. Los IValueConverter interfaz se compone de dos métodos, denominados Convertir y ConvertBack, con parámetros idénticos. Puede hacer que la clase como generalizada o tan especializado como desee:

utilizando Sistema;
utilizando System.Globalization;
utilizando Xamarin.Forms;

```
espacio de nombres Xamarin.FormsBook.Toolkit
{
    clase pública IntToBoolConverter : IValueConverter
    {
        objeto público Convertir( objeto valor, Tipo tipo de objetivo,
                                  objeto parámetro, CultureInfo cultura)
        {
            regreso ( Ent ) Valor = 0;
        }

        objeto público ConvertBack ( objeto valor, Tipo tipo de objetivo,
                                    objeto parámetro, CultureInfo cultura)
        {
            regreso ( bool ) Valor? 1: 0;
        }
    }
}
```

Al incluir esta clase en un enlace de datos, y verá cómo hacer que poco -la Convertir método se llama cuando un valor pasa desde el origen al destino.

los valor argumento para Convertir es el valor de la fuente de enlace de datos a convertir. Puedes usar GetType para determinar su tipo, o puedes asumir que siempre es un tipo particular. En este ejemplo, la valor argumento se supone que es de tipo Ent, por lo que la fundición a una Ent no lanzará una excepción. convertidores de valores más sofisticados pueden realizar más comprobaciones de validez.

los tipo de objetivo es el tipo de la propiedad de destino de enlace de datos. convertidores de valores versátiles pueden utilizar este argumento para adaptar la conversión para los diferentes tipos de destino. los Convertir método debe devolver una

objeto o valor que coincide con esta tipo de objetivo. Este particular Convertir método supone que objetivo-Tipo es bool.

los parámetro argumento es un parámetro de conversión opcional que se puede especificar como un alojamiento a la Unión clase. (Vas a ver un ejemplo en el capítulo 18, "MVVM.")

Por último, si es necesario realizar una conversión específica de la cultura, el último argumento es el CultureInfo objeto que se debe utilizar.

El cuerpo de este particular Convertir método supone que valor es una Ent, y el método devuelve una bool es decir cierto si ese entero es distinto de cero.

los ConvertBack método se llama sólo para TwoWay o OneWayToSource Enlaces. Para el Esta-vertBack método, el valor argumento es el valor de la diana y la tipo de objetivo argumento es en realidad el tipo de la propiedad de origen. Si sabe que la ConvertBack Nunca método será llamado, puede simplemente ignorar todos los argumentos y vuelta nulo o 0 de la misma. Con algunos convertidores de valores, la aplicación de una ConvertBack cuerpo es prácticamente imposible, pero a veces es bastante simple (como en este caso).

Cuando se utiliza un convertidor de valor en el código, se establece una instancia del convertidor a la Convertidor propiedad de Unión. Usted puede opcionalmente pasar un argumento al convertidor de valores mediante el establecimiento de la Esta-verterParameter propiedad de Unión.

Si la unión tiene también una StringFormat, el valor que se devuelve por el convertidor de valores es el valor que tiene el formato como una cadena.

En general, en un archivo XAML que querrá crear una instancia del convertidor de valores en una recursos Diccionario y luego hacer referencia a ella en el Unión expresión mediante el uso de StaticResource. El convertidor de valores no debe mantener el estado y por lo tanto se puede compartir entre varios enlaces.

Aquí está la ButtonEnabler programa que utiliza el convertidor de valores:

```
< Página de contenido xmlns = "http://xamarin.com/schemas/2014/forms" 
    xmlns: x = "http://schemas.microsoft.com/winfx/2009/xaml" 
    xmlns: kit de herramientas = 
        "clr-espacio de nombres: Xamarin.FormsBook.Toolkit; montaje = Xamarin.FormsBook.Toolkit" 
    x: Class = "ButtonEnabler.ButtonEnablerPage" 
    Relleno = "10, 50, 10, 0" >

    < ContentPage.Resources >
        < ResourceDictionary >
            < kit de herramientas: IntToBoolConverter x: Key = "intToBool" />
        </ ResourceDictionary >
    </ ContentPage.Resources >

    < StackLayout Espaciado = "20" >
        < Entrada x: Nombre = "entrada" 
            Texto = " 
            marcador de posición = " texto al botón de activación " />
```

```
< Botón Texto = "Guardar o enviar (o algo así)" 
    Tamaño de fuente = "Medio" 
    HorizontalOptions = "Centrar" 
    Está habilitado = "{Binding Fuente = {x: entrada Referencia}}, 
        Path = text.length, 
        Convertidor = {} StaticResource intToBool" />

</ StackLayout >
</ Pagina de contenido >
```

los IntToBoolConverter se crea una instancia en el recursos diccionario y que se hace referencia como extensión de marcado anidado en el Unión que se establece en el Está habilitado propiedad de la Botón.

Observe que el Texto la propiedad es inicializado explícitamente en el Entrada etiqueta a una cadena vacía. Por defecto, el Texto propiedad es nulo, lo que significa que la unión Camino ajuste de text.length no da lugar a un valor válido.

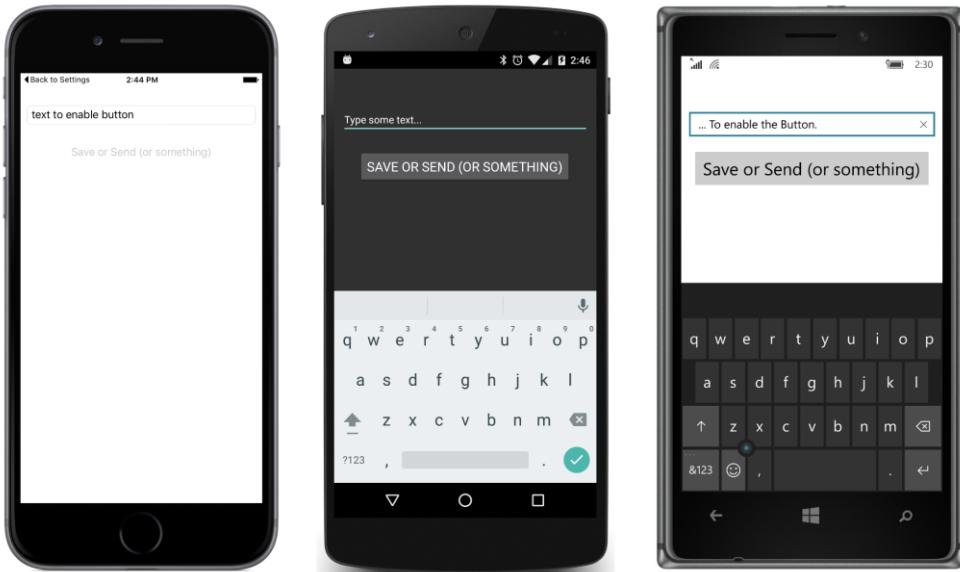
Puede que recuerde de los capítulos anteriores que una clase en el **Xamarin.FormsBook.Toolkit** biblioteca que se hace referencia sólo en XAML no es suficiente para establecer un enlace desde la aplicación a la biblioteca. Por esa razón, el Aplicación constructor de **ButtonEnabler** llamadas Toolkit.Init:

```
público clase Aplicación : Solicitud
{
    público App ()
    {
        Xamarin.FormsBook.Toolkit .En eso();

        MainPage = nuevo ButtonEnablerPage ();
    }
    ...
}
```

código similar aparece en todos los programas de este capítulo que utilizan el **Xamarin.FormsBook.Toolkit** biblioteca.

Las imágenes confirman que la Botón No está permitido a menos que el Entrada contiene un texto:



Si está utilizando sólo una instancia de un convertidor de valores, no es necesario almacenarlo en el recurso diccionario. Usted puede crear una instancia que justo en el Unión etiqueta con el uso de etiquetas de propiedad de elementos de la propiedad de destino y para el Convertidor propiedad de Unión:

```
<Botón Texto = "Guardar o enviar (o algo así)" >
    Tamaño de fuente = "Grande"
    HorizontalOptions = "Centrar" >
<Button.IsEnabled>
    <Unión Fuente = "(X: entrada de referencia)" >
        Camino = "text.length" >
        <Binding.Converter>
            <kit de herramientas: Int.ToBooleanConverter />
        </Binding.Converter>
    </Unión>
</Button.IsEnabled>
</Botón>
```

A veces es conveniente para un convertidor de valores para definir un par de propiedades simples. Por ejemplo, supongamos que desea mostrar algún texto para las dos configuraciones de una Cambiar pero usted no quiere utilizar el "verdadero" y "falso", y que no quiere alternativas codificar directamente en el convertidor de valores. Aquí está un

BoolToStringConverter con un par de propiedades públicas por dos cadenas de texto:

```
espacio de nombres Xamarin.FormsBook.Toolkit
{
    clase pública BoolToStringConverter : IValueConverter
    {
        public string TrueText { conjunto ; obtener ; }

        public string FalseText { conjunto ; obtener ; }
```

```

    objeto público Convertir( objeto valor, Tipo tipo de objetivo,
                           objeto parámetro, CultureInfo cultura)
    {
        regreso ( bool ) Valor? TrueText: FalseText;
    }

    objeto público ConvertBack ( objeto valor, Tipo tipo de objetivo,
                               objeto parámetro, CultureInfo cultura)
    {
        falso retorno ;
    }
}
}

```

El cuerpo de la Convertir método es trivial: sólo se selecciona entre los dos cadenas basadas en el booleano valor argumento.

Un convertidor de valores similares convierte un valor booleano a uno de dos colores:

```

espacio de nombres Xamarin.FormsBook.Toolkit
{
    clase pública BoolToColorConverter : IValueConverter
    {
        público Color Color verdadero { conjunto ; obtener ; }

        público Color falso color { conjunto ; obtener ; }

        objeto público Convertir( objeto valor, Tipo tipo de objetivo,
                               objeto parámetro, CultureInfo cultura)
        {
            regreso ( bool ) Valor? True Color: falso color;
        }

        objeto público ConvertBack ( objeto valor, Tipo tipo de objetivo,
                               objeto parámetro, CultureInfo cultura)
        {
            falso retorno ;
        }
    }
}

```

los **SwitchText** crea una instancia del programa **BoolToStringConverter** convertidor de dos veces por dos pares diferentes de cadenas: una vez en el recursos diccionario y, a continuación, dentro de **BindingConverter** etiquetas de propiedad de elementos. Dos propiedades de la final Etiqueta se someten a la **BoolToStringConverter**

y el **BoolToColorConverter** basado en el mismo **IsToggled** propiedad de la Cambiar:

```

< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
  xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
  xmlns: kit de herramientas =
  "clr-espacio de nombres: Xamarin.FormsBook.Toolkit; montaje = Xamarin.FormsBook.Toolkit "
  x: Class = " SwitchText.SwitchTextPage "
  Relleno = " 10, 0 " >

```

```
< ContentPage.Resources >
    < ResourceDictionary >
        < kit de herramientas: BoolToStringConverter x: Key = " boolToString "
            TrueText = " Vamos a hacerlo "
            FalseText = " Ahora no " />

        < Estilo Tipo de objetivo = " Etiqueta " >
            < Setter Propiedad = " Tamaño de fuente " Valor = " Medio " />
            < Setter Propiedad = " VerticalOptions " Valor = " Centrar " />
        </ Estilo >
    </ ResourceDictionary >
</ ContentPage.Resources >

< StackLayout >
    <! - Primera conexión con el texto. ->
    < StackLayout Orientación = " Horizontal "
        VerticalOptions = " CenterAndExpand " >
        < Etiqueta Texto = " ¿Aprende más? " />

        < Cambiar x: Nombre = " switch1 "
            VerticalOptions = " Centrar " />

        < Etiqueta Texto = " {Binding Fuente = {x: switch1 Referencia},
            Path = IsToggled,
            Convertidor = {} StaticResource boolToString "
            HorizontalOptions = " FillAndExpand " />
    </ StackLayout >

    <! - En segundo lugar Interruptor con texto. ->
    < StackLayout Orientación = " Horizontal "
        VerticalOptions = " CenterAndExpand " >
        < Etiqueta Texto = " ¿Suscribir? " />

        < Cambiar x: Nombre = " switch2 "
            VerticalOptions = " Centrar " />

        < Etiqueta Texto = " {Binding Fuente = {x: switch2 Referencia},
            Path = IsToggled,
            Convertidor = {} StaticResource boolToString "
            HorizontalOptions = " FillAndExpand " />
    </ StackLayout >

    <! - En tercer lugar Interruptor con texto y color. ->
    < StackLayout Orientación = " Horizontal "
        VerticalOptions = " CenterAndExpand " >
        < Etiqueta Texto = " ¿Dejar página? " />

        < Cambiar x: Nombre = " switch3 "
            VerticalOptions = " Centrar " />

        < Etiqueta HorizontalOptions = " FillAndExpand " >
            < label.text >
                < Unión Fuente = " {x: switch3 Referencia} "
                    Camino = " IsToggled " >
            </ label.text >
        </ Etiqueta >
    </ StackLayout >
```

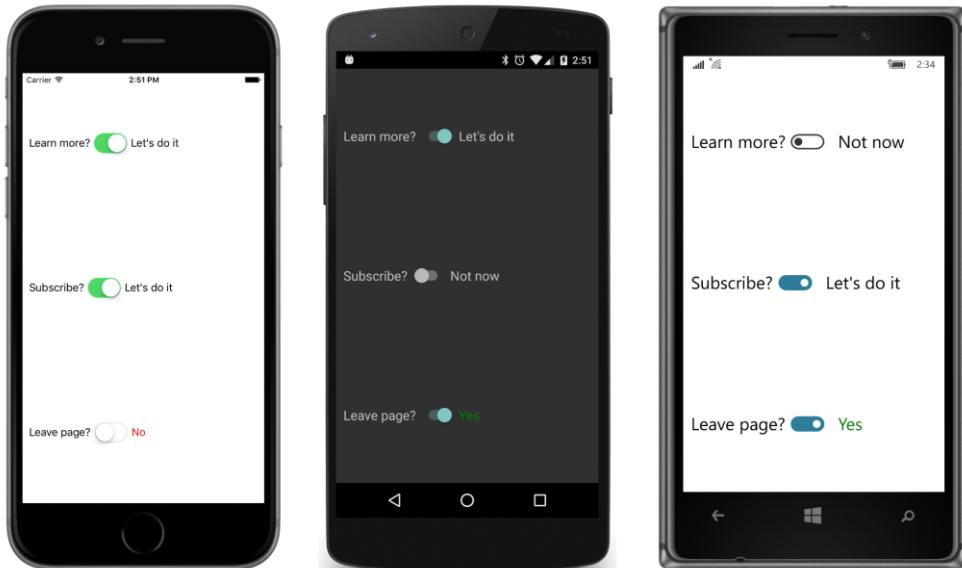
```

< Binding.Converter >
    < kit de herramientas: BoolToStringConverter TrueText = " Sí "
        FalseText = " No " />
</ Binding.Converter >
</ Unión >
</ label.text >

< Label.TextColor >
    < Unión Fuente = "(X: switch3 Referencia)" 
        Camino = " IsToggled " >
< Binding.Converter >
    < kit de herramientas: BoolToColorConverter Color verdadero = " Verde "
        falso color = " rojo " />
</ Binding.Converter >
</ Unión >
</ Label.TextColor >
</ Etiqueta >
</ StackLayout >
</ StackLayout >
</ Página de contenido >

```

Con los dos convertidores de enlace bastante trivial, la Cambiar Ahora puede mostrar el texto que desea para los dos estados y puede colorear el texto con colores personalizados:



Ahora que usted ha visto una `BoolToStringConverter` y una `BoolToColorConverter`, se puede generalizar la técnica a objetos de cualquier tipo? Aquí es un genérico `BoolToObjectConverter`. También en el `Xamarin.FormsBook.Toolkit` biblioteca:

```

pública clase BoolToObjectConverter <T>: IValueConverter
{
    pública T {TrueObject conjunto ; obtener ;}
}

```

```

público T {FalseObject conjunto ; obtener ;}

objeto público Convertir( objeto valor, Tipo tipo de objetivo,
                           objeto parámetro, CultureInfo cultura)
{
    regreso ( bool ) Valor? esta .TrueObject: esta .FalseObject;
}

objeto público ConvertBack ( objeto valor, Tipo tipo de objetivo,
                           objeto parámetro, CultureInfo cultura)
{
    regreso ((T) de valor) .equals ( esta .TrueObject);
}
}

```

El siguiente ejemplo utiliza esta clase.

Fijaciones y vistas personalizadas

En el capítulo 15, "La interfaz interactiva", que vio una vista personalizada denominada Caja. Este punto de vista define una Texto propiedad para establecer el texto de la Caja así como una Tamano de fuente propiedad. También podría haber definido todas las otras propiedades- relacionadas con el texto TextColor, FontAttributes, y Familia tipográfica -pero no lo hizo, sobre todo debido a los trabajos. Cada propiedad requiere una BindableProperty definición, una definición de propiedad CLR, y un controlador de la propiedad cambiado que transfiere el nuevo valor de la propiedad a la Etiqueta vistas que comprenden las representaciones visuales de la Caja.

Los enlaces de datos pueden ayudar a simplificar este proceso para algunas propiedades mediante la eliminación de los manipuladores PropertyChanged.

Aquí está el archivo de código subyacente para una nueva versión de Caja llamado NewCheckBox.

Al igual que la clase anterior, es parte de la **Xamarin.FormsBook.Toolkit** biblioteca. El archivo se ha reorganizado un poco para que cada uno BindableProperty definición está emparejado con su correspondiente definición de propiedad CLR. Es posible que prefiera este tipo de organización de código fuente de las propiedades, o tal vez no.

```

espacio de nombres Xamarin.FormsBook.Toolkit
{
    público clase parcial NewCheckBox : ContentView
    {
        evento público Controlador de eventos < bool > CheckedChanged;

        público NewCheckBox ()
        {
            InitializeComponent ();
        }

        // propiedad Text.
        sólo lectura estática pública BindableProperty TextProperty =
            BindableProperty .Crear(
                "Texto",
                tipo de ( cuerda ),

```

```
    tipo de ( NewCheckBox ),
    nulo );

public string Texto
{
    conjunto {EstablecerValor (TextProperty, valor ); }
    obtener { regreso ( cuenda ) GetValue (TextProperty); }
}

// propiedad TextColor.

sólo lectura estática pública BindableProperty TextColorProperty =
    BindableProperty .Crear(
        "Color de texto",
        tipo de ( Color ),
        tipo de ( NewCheckBox ),
        Color .Defecto);

público Color Color de texto
{
    conjunto {EstablecerValor (TextColorProperty, valor ); }
    obtener { regreso ( Color ) GetValue (TextColorProperty); }
}

// propiedad Tamaño de Letra.

sólo lectura estática pública BindableProperty FontSizeProperty =
    BindableProperty .Crear(
        "Tamaño de fuente",
        tipo de ( doble ),
        tipo de ( NewCheckBox ),
        Dispositivo .GetNamedSize ( NamedSize .Defecto, tipo de ( Etiqueta )));

[ TypeConverter ( tipo de ( FontSizeConverter ))]

pública doble Tamaño de fuente
{
    conjunto {EstablecerValor (FontSizeProperty, valor ); }
    obtener { regreso ( doble ) GetValue (FontSizeProperty); }
}

// FontAttributes propiedad.

sólo lectura estática pública BindableProperty FontAttributesProperty =
    BindableProperty .Crear(
        "FontAttributes",
        tipo de ( FontAttributes ),
        tipo de ( NewCheckBox ),
        FontAttributes .Ninguna);

público FontAttributes FontAttributes
{
    conjunto {EstablecerValor (FontAttributesProperty, valor ); }
    obtener { regreso ( FontAttributes ) GetValue (FontAttributesProperty); }
}

// propiedad IsChecked.

sólo lectura estática pública BindableProperty IsCheckedProperty =
```

```

BindableProperty .Crear(
    "Está chequeado" ,
    tipo de ( bool ),
    tipo de ( NewCheckBox ),
    falso ,
    PropertyChanged: (enlazable, oldValue, newValue) =>
{
    // desencadenar el evento.
    NewCheckBox casilla de verificación = ( NewCheckBox ) Enlazable;
    Controlador de eventos < bool > EventHandler = checkbox.CheckedChanged;
    Si (EventHandler1 = nulo )
    {
        eventHandler (casilla de verificación, ( bool )nuevo valor);
    }
});

public bool Está chequeado
{
    conjunto {EstablecerValor (IsCheckedProperty, valor); }
    obtener {regreso ( bool ) GetValue (IsCheckedProperty); }
}

// manipulador TapGestureRecognizer.
vacío OnCheckBoxTapped ( objeto remitente, EventArgs args)
{
    IsChecked = IsChecked!;
}
}
}

```

Además de la anterior Texto y Tamaño de fuente propiedades, este archivo de código ahora también define Color de texto y FontAttributes propiedades. Sin embargo, el único controlador de la propiedad es cambiado para el Está chequeado manejador para disparar el CheckedChanged evento. Todo lo demás es manejado por enlaces de datos en el archivo XAML:

```

< ContentView xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    xmlns: kit de herramientas = " CLR-espacio de nombres: Xamarin.FormsBook.Toolkit "
    x: Class = " Xamarin.FormsBook.Toolkit.NewCheckBox "
    x: Nombre = " caja " >

    < StackLayout Orientación = " Horizontal "
        BindingContext = " {X: Referencia casilla} " >

        < Etiqueta x: Nombre = " box_label " Texto = " & # X2610; "
            Color de texto = " { La unión TextColor } "
            Tamaño de fuente = " {Binding} Tamaño de Letra " >

            < label.text >
                < Unión Camino = " Está chequeado " >
                    < Binding Converter >
                        < kit de herramientas: BoolToStringConverter TrueText = " & # X2611; "
                            FalseText = " & # X2610; " />
                    </ Binding Converter >
                </ Unión >
            </ label.text >
        
```

```

</ Etiqueta >

< Etiqueta x: Nombre = "textLabel" Texto = "{Binding Path = texto}" 
    Color de texto = "0 La unión TextColor" 
    Tamaño de fuente = "{Binding} Tamaño de Letra" 
    FontAttributes = "0 La unión FontAttributes" />

</ StackLayout >

< ContentView.GestureRecognizers >
    < TapGestureRecognizer aprovechado = "OnCheckBoxTapped" />
</ ContentView.GestureRecognizers >

</ ContentView >

```

El elemento raíz se le da un nombre de caja, y el StackLayout establece que a medida que su Unión-Contexto. Todos los enlaces de datos dentro de ese StackLayout a continuación, puede hacer referencia a las propiedades definidas por el archivo de código subyacente. El primero Etiqueta que muestra el cuadro tiene su Color de texto y Tamaño de fuente propiedades unidos a los valores de las propiedades subyacentes, mientras que el Texto la propiedad está dirigido por una unión que utiliza una BoolToStringConverter para mostrar una caja vacía o una casilla marcada en base a la Está chequeado propiedad. El segundo Etiqueta es más directo: el Texto, TextColor, Tamaño de Letra, y FontAttributes Todas ellas están ligadas a las propiedades correspondientes definidas en el archivo de código subyacente.

Si va a crear varias vistas personalizadas que incluyen Texto elementos y que necesita definiciones de todas las propiedades relacionadas con el texto, es probable que desee crear primero una clase de sólo código (el nombre Vista personalizada-Base, por ejemplo) que se deriva de ContentView e incluye solamente aquellas definiciones de propiedad basados en texto. A continuación, puede derivar otras clases de CustomViewBase y tiene Texto y todas las propiedades textrelated fácilmente disponibles.

Vamos a escribir un pequeño programa llamado **NewCheckBoxDemo** que demuestra la NewCheckBox ver. Al igual que el anterior **CheckBoxDemo** programa, estas casillas de verificación controlan el formato de negrita y cursiva de un párrafo de texto. Pero para demostrar las nuevas propiedades, estas casillas se dan colores y atributos de fuente, y para demostrar la BoolToObjectConverter, una de las casillas de verificación controla la alineación horizontal de dicho párrafo:

```

< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    xmlns: kit de herramientas =
        "clr-espacio de nombres: Xamarin.FormsBook.Toolkit; montaje = Xamarin.FormsBook.Toolkit"
    x: Class = "NewCheckBoxDemo.NewCheckBoxDemoPage" >

    < StackLayout Relleno = "10, 0" >
        < StackLayout HorizontalOptions = "Centrar" 
            VerticalOptions = "CenterAndExpand" >

            < StackLayout.Resources >
                < ResourceDictionary >
                    < Estilo Tipo de objetivo = "kit de herramientas: NewCheckBox" >
                        < Setter Propiedad = "Tamaño de fuente" Valor = "Grande" />
                    </ Estilo >
                </ ResourceDictionary >
            </ StackLayout.Resources >

```

```

< kit de herramientas: NewCheckBox Texto = " Ítálico "
    Color de texto = " Agua "
    Tamaño de fuente = " Grande "
    FontAttributes = " Ítálico "
    CheckedChanged = " OnItalicCheckBoxChanged " />

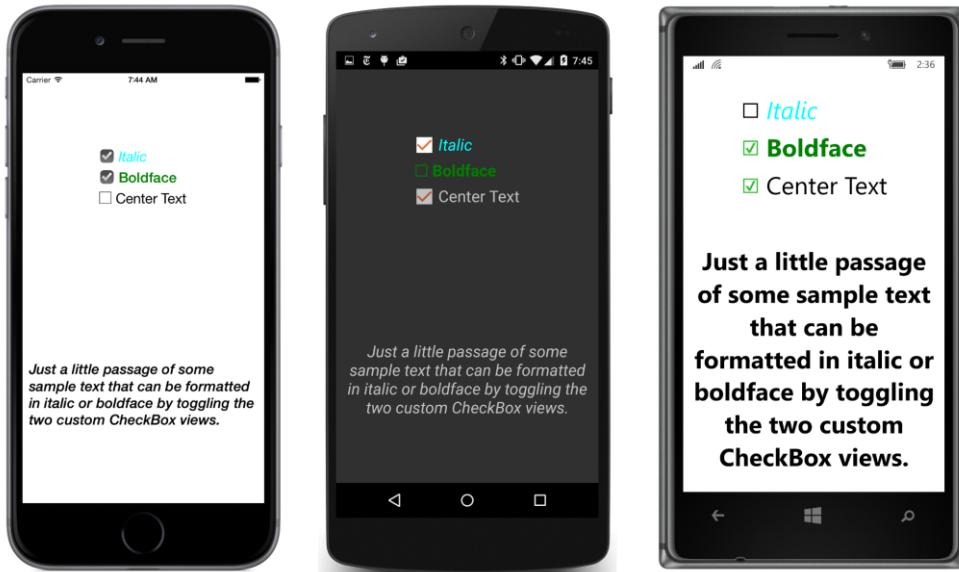
< kit de herramientas: NewCheckBox Texto = " negrita "
    Tamaño de fuente = " Grande "
    Color de texto = " Verde "
    FontAttributes = " Negrita "
    CheckedChanged = " OnBoldCheckBoxChanged " />

< kit de herramientas: NewCheckBox x: Nombre = " centerCheckBox "
    Texto = " Centrar texto " />
</ StackLayout >

< Etiqueta x: Nombre = " etiqueta "
    Texto =
    " Sólo un pequeño paso de texto de ejemplo que se pueden formatear
en cursiva o negrita alterando los dos puntos de vista CheckBox personalizado. "
    Tamaño de fuente = " Grande "
    VerticalOptions = " CenterAndExpand " >
    < Label.HorizontalTextAlignment >
        < Unión Fuente = "(X: Referencia centerCheckBox) "
            Camino = " Está chequeado " >
            < Binding.Converter >
                < kit de herramientas: BoolToObjectConverter x: TypeArguments = " Alineación del texto "
                    TrueObject = " Centrar "
                    FalseObject = " comienzo " />
                </ Binding.Converter >
            </ Unión >
        </ Label.HorizontalTextAlignment >
    </ Etiqueta >
</ StackLayout >
</ Pagina de contenido >

```

Observe la `BoolToObjectConverter` Entre los `Binding.Converter` las etiquetas. Debido a que es una clase genérica, se requiere una `x: TypeArguments` atributo que indica el tipo de la `TrueObject` y `FalseObject` propiedades y el tipo del valor de retorno de la `Convertir` método. Ambos `TrueObject` y `FalseObject` se establecen para los miembros de la `Alineación del texto` enumeración, y el convertidor selecciona uno que se establece en la `HorizontalTextAlignment` propiedad de la Etiqueta, como las siguientes imágenes demuestran:



Sin embargo, este programa todavía necesita un archivo de código subyacente para gestionar la aplicación de la cursiva y negrita atribuye al bloque de texto. Estos métodos son idénticos a los de la primera **CheckBoxDemo** programa:

```

público clase parcial NewCheckBoxDemoPage : Página de contenido
{
    público NewCheckBoxDemoPage ()
    {
        InitializeComponent ();
    }

    vacio OnItalicCheckBoxChanged ( objeto remitente, bool esta chequeado )
    {
        Si (esta chequeado)
        {
            label.FontAttributes |= FontAttributes .Itálico;
        }
        más
        {
            label.FontAttributes &= ~ FontAttributes .Itálico;
        }
    }

    vacio OnBoldCheckBoxChanged ( objeto remitente, bool esta chequeado )
    {
        Si (esta chequeado)
        {
            label.FontAttributes |= FontAttributes .Negrita;
        }
        más
        {
    }
}

```

```
label.FontAttributes & = ~FontAttributes.Negrita;  
}  
}  
}
```

Xamarin.Forms no admite un “multi-unión” que podrían permitir que varios orígenes de enlace que se combinan para cambiar una sola diana de unión. Encuadernaciones pueden hacer mucho, pero sin algún apoyo código adicional, que no pueden hacer todo.

Todavía hay un papel para el código.

capítulo 17

El dominio de la cuadrícula

los Cuadrícula es un poderoso mecanismo de diseño que organiza sus hijos en filas y columnas de celdas. Al principio, la Cuadrícula parece asemejarse a la de HTML mesa, pero hay una distinción muy importante: El HTML mesa está diseñado para fines de presentación, mientras que el Cuadrícula es el único diseño. No existe el concepto de un encabezamiento en una Cuadrícula, por ejemplo, y ninguna característica incorporada para dibujar cuadros alrededor de las células o para separar filas y columnas con líneas divisorias. Los puntos fuertes de la Cuadrícula son en la especificación de las dimensiones de celda con tres opciones de configuración de altura y anchura.

Como hemos visto, la StackLayout es ideal para unidimensionales colecciones de los niños. Aunque es posible que un nido StackLayout Dentro de un StackLayout para acomodar una segunda dimensión y imitar una mesa, a menudo el resultado puede exhibir problemas de alineación. Los Cuadrícula, sin embargo, está diseñado específicamente para dos matrices unidimensionales de niños. Como verá hacia el final de este capítulo, el Cuadrícula También puede ser muy útil para administrar presentaciones que se adaptan a ambos modos vertical y horizontal.

La red básica

UN Cuadrícula se puede definir y se llena con los niños, ya sea en código o XAML, pero el enfoque XAML es más fácil y más clara, y por lo tanto de lejos el más común.

La cuadrícula en XAML

Cuando se define en XAML, una Cuadrícula casi siempre tiene un número fijo de filas y columnas. Los Cuadrícula definición generalmente comienza con dos propiedades importantes, denominadas (RowDefinitions que es una colección de RowDefinition objetos) y (ColumnDefinitions una colección de ColumnDefinition objetos). Estas colecciones contienen una RowDefinition para cada fila de la Cuadrícula y uno ColumnDefinition para cada columna, y se definen las características de fila y columna de la Cuadrícula.

UN Cuadrícula puede consistir en una única fila o columna única (en cuyo caso no se necesita una de las dos definiciones colecciones), o incluso de una sola célula.

RowDefinition tiene un Altura propiedad de tipo GridLength, y ColumnDefinition tiene un Anchura propiedad, también de tipo GridLength. Los GridLength estructura especifica una altura de la fila o un ancho de columna en términos de la GridUnitType enumeración, que tiene tres miembros:

- Absoluto anchura o altura -el es un valor en unidades independientes del dispositivo (un número en XAML)
- Auto anchura o altura -el se autosized basan en los contenidos de la celda ("Auto" en XAML)
- Estrella -leftover anchura o altura se asigna proporcionalmente (un número con "*" en XAML)

Aquí está la primera mitad del archivo XAML en el **SimpleGridDemo** proyecto:

```
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " SimpleGridDemo.SimpleGridDemoPage " >

    < ContentPage.Padding >
        < OnPlatform x: TypeArguments = " Espesor "
            iOS = " 0, 20, 0, 0 " />
    </ ContentPage.Padding >

    < Cuadrícula >
        < Grid.RowDefinitions >
            < RowDefinition Altura = " Auto " />
            < RowDefinition Altura = " 100 " />
            < RowDefinition Altura = " 2 ** " />
            < RowDefinition Altura = " 1 ** " />
        </ Grid.RowDefinitions >

        < Grid.ColumnDefinitions >
            < ColumnDefinition Anchura = " * " />
            < ColumnDefinition Anchura = " * " />
        </ Grid.ColumnDefinitions >

        ...
    </ Cuadrícula >
</ Pagina de contenido >
```

Esta Cuadrícula tiene cuatro filas y dos columnas. La altura de la primera fila es "Auto", justificado la altura se calcula en base a la altura máxima de todos los elementos que ocupan la primera fila. La segunda fila es de 100 unidades independientes del dispositivo de altura.

Los dos Altura ajustes con "*" (pronunciado "estrella") requieren una explicación adicional: Este particular Cuadrícula tiene una altura total que es la altura de la página menos el Relleno establecer en IOS. Internamente, el Cuadrícula determina la altura de la primera fila basado en el contenido de esa fila, y se sabe que la altura de la segunda fila es 100. Se resta estos dos alturas de su propia altura y asigna la altura restante proporcionalmente entre las filas tercera y cuarta basados sobre el número en la configuración de estrella. La tercera fila es el doble de la altura de la cuarta fila.

Los dos ColumnDefinition Los objetos ubicados tanto en el Anchura igual a " * ", que es lo mismo que " 1 ** ", que significa que la anchura de la pantalla se divide por igual entre las dos columnas.

Usted recordará del capítulo 14, "disposición absoluta", que la **AbsoluteLayout** clase define dos propiedades enlazables adjunto y cuatro estático Conjunto y Obtener métodos que permiten a un programa para especificar la posición y el tamaño de un niño de la **AbsoluteLayout** en clave o XAML.

los Cuadrícula es bastante similar. los Cuadrícula clase define cuatro propiedades enlazables adjunta, para la especificación de la celda o celdas que un niño de la Cuadrícula ocupa:

- Grid.RowProperty -la fila de base cero; valor por defecto es 0

- Grid.ColumnProperty -la columna de base cero; valor por defecto es 0
- Grid.RowSpanProperty -el número de filas que el niño palmos; valor predeterminado es 1
- Grid.ColumnSpanProperty -El número de columnas que el niño palmos; valor predeterminado es 1

Todos los cuatro propiedades se definen para ser de tipo En t.

Por ejemplo, para especificar en el código que una Cuadrícula niño llamado ver reside en una fila y columna en particular, puede llamar a:

```
view.SetValue ( Cuadricula .RowProperty, 2);
view.SetValue ( Cuadricula .ColumnProperty, 1);
```

Esos son los números de fila y columna de base cero, por lo que el niño es asignado a la tercera fila y la segunda columna.

los Cuadricula clase también define ocho métodos estáticos para simplificar la configuración y obtener una de estas propiedades en código:

- Grid.SetRow y Grid.GetRow
- Grid.SetColumn y Grid.GetColumn
- Grid.SetRowSpan y Grid.GetRowSpan
- Grid.SetColumnSpan y Grid.GetColumnSpan

Aquí está el equivalente de los dos Valor ajustado llamadas que acaba de ver:

```
Cuadricula .SetRow (vista, 2);
Cuadricula .SetColumn (vista, 1);
```

Como se vio en relación con AbsoluteLayout, tales estática Conjunto y Obtener métodos se implementan con Valor ajustado y GetValue pide a la hija de Cuadricula. Por ejemplo, he aquí cómo SetRow Es muy probable definido dentro de la Cuadricula clase:

```
public hoyo estatico SetRow ( bindableObject enlazable, En t valor)
{
    bindable.SetValue ( Cuadricula .RowProperty, valor);
}
```

No se puede llamar a estos métodos en XAML, así que en vez que utilice los siguientes atributos para definir las propiedades enlazables unidos en un niño de la Cuadricula:

- Grid.Row
- Grid.Column
- Grid.RowSpan
- Grid.ColumnSpan

Estos atributos XAML en realidad no son definidos por la Cuadrícula clase, pero el analizador XAML sabe que debe hacer referencia a las propiedades enlazables adjuntos asociados definidos por Cuadrícula.

No es necesario configurar todas estas propiedades en todos los niños de la Cuadrícula. Si el niño ocupa sólo una célula, entonces no establezca Grid.RowSpan o Grid.ColumnSpan porque el valor predeterminado es 1. El Grid.Row y Grid.Column propiedades tienen un valor por defecto de 0, por lo que no es necesario establecer los valores si el niño ocupa la primera fila o la primera columna. Sin embargo, por motivos de claridad, el código de este libro generalmente mostrará la configuración de estas dos propiedades. Para ahorrar espacio, a menudo estos atributos van a aparecer en la misma línea en la lista de XAML.

Aquí está el archivo XAML completa para SimpleGridDemo:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " SimpleGridDemo.SimpleGridDemoPage " >

    < ContentPage.Padding >
        < OnPlatform x: TypeArguments = " Espesor "
            iOS = " 0, 20, 0, 0 " />
    </ ContentPage.Padding >

    < Cuadrícula >
        < Grid.RowDefinitions >
            < RowDefinition Altura = " Auto " />
            < RowDefinition Altura = " 100 " />
            < RowDefinition Altura = " 2 * " />
            < RowDefinition Altura = " 1 * " />
        </ Grid.RowDefinitions >

        < Grid.ColumnDefinitions >
            < ColumnDefinition Anchura = " * " />
            < ColumnDefinition Anchura = " * " />
        </ Grid.ColumnDefinitions >

        < Etiqueta Texto = " rejilla de demostración "
            Grid.Row = " 0 " Grid.Column = " 0 "
            Tamaño de fuente = " Grande "
            HorizontalOptions = " Fin " />

        < Etiqueta Texto = " Demostración de la cuadrícula "
            Grid.Row = " 0 " Grid.Column = " 1 "
            Tamaño de fuente = " Pequeña "
            HorizontalOptions = " Fin "
            VerticalOptions = " Fin " />

        < Imagen Color de fondo = " gris "
            Grid.Row = " 1 " Grid.Column = " 0 " Grid.ColumnSpan = " 2 " >
            < Fuente de imagen >
                < OnPlatform x: TypeArguments = " Fuente de imagen "
                    iOS = " Icono-60.png "
                    Androide = " icon.png "
                    WinPhone = " Activos / StoreLogo.png " />
```

```

</ Fuente de imagen >
</ Imagen >

< BoxView Color = " Verde "
    Grid.Row = " 2 " Grid.Column = " 0 " />

< BoxView Color = " rojo "
    Grid.Row = " 2 " Grid.Column = " 1 " Grid.RowSpan = " 2 " />

< BoxView Color = " Azul "
    Opacidad = " 0.5 "
    Grid.Row = " 3 " Grid.Column = " 0 " Grid.ColumnSpan = " 2 " />

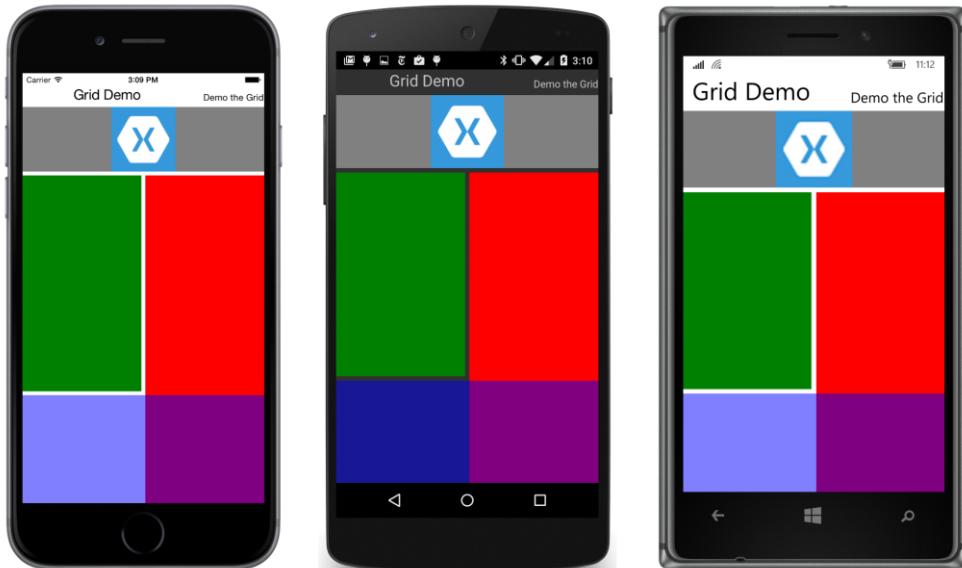
</ Cuadricula >
</ Pagina de contenido >

```

Dos Etiqueta elementos con diferentes Tamaño de fuente ajustes ocupan las dos columnas de la primera fila. La altura de esa fila se rige por el elemento más alto. ajustes de HorizontalOptions y verticalmente calOptions puede posicionar un niño dentro de la celda.

La segunda fila tiene una altura de 100 unidades independientes del dispositivo. Esa fila está ocupada por una Imagen elemento que muestra un icono de aplicación con un fondo gris. los Imagen elemento se extiende por ambas columnas de esa fila.

Las dos filas inferiores están ocupadas por tres BoxView elementos, uno que abarca dos filas, y otra que se extiende por dos columnas, y éstas se solapan en la celda inferior derecha:



Las imágenes confirman que la primera fila está dimensionada para la altura de la gran Etiqueta; la segunda fila es de 100 unidades independientes del dispositivo de alto; y la tercera y cuarta filas ocupan todo el espacio restante. La tercera fila es dos veces tan alto como el cuarto. Las dos columnas son iguales en anchura y dividen el entero Cuadricula

a la mitad. El rojo y el azul BoxView elementos se superponen en la celda inferior derecha, pero el azul BoxView obviamente, se sienta encima de la roja, ya que tiene una Opacidad ajuste de 0,5 y el resultado es de color púrpura.

La mitad izquierda del azul semitransparente BoxView es más ligero en el dispositivo móvil iPhone y Windows 10 que en el teléfono Android por lo fondo blanco.

Como se puede ver, los niños de la Cuadrícula puede compartir celdas. El orden en que los niños aparecen en el archivo XAML es el orden en que los niños se ponen en el Cuadrícula, con los niños más tarde, aparentemente se sienta encima de (y oscurecimiento) hijos anteriores.

Se dará cuenta de que un pequeño hueco parece separar las filas y columnas donde el fondo se asoma a través. Esto se rige por dos Cuadrícula propiedades:

- Distancia entre filas valor -default de 6
- ColumnSpacing valor -default de 6

Puede establecer estas propiedades a 0 si desea cerrar ese espacio, y se puede establecer la Fondo-Color propiedad de la Cuadrícula si desea que el color mirando a través de que sea algo diferente. También puede agregar espacio en el interior de la Cuadrícula en todo su perímetro con una Relleno el establecimiento de la Cuadrícula.

Ahora ha sido introducido a todas las propiedades públicas y métodos definidos por Cuadrícula.

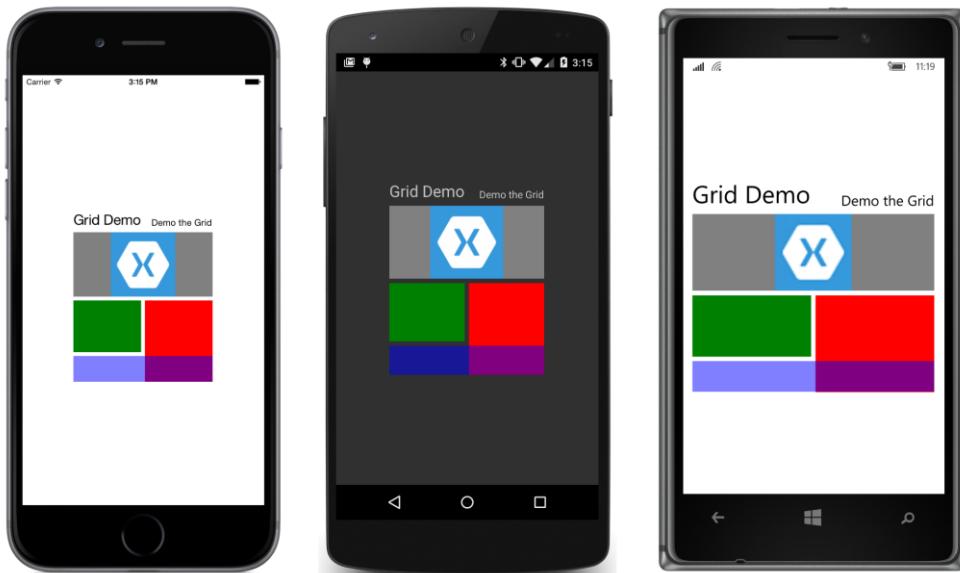
Antes de continuar, vamos a realizar un par de experimentos con **SimpleGridDemo**. En primer lugar, comentario o eliminar la totalidad RowDefinitions y ColumnDefinitions sección cerca de la parte superior de la Cuadrícula, y luego volver a implementar el programa. Esto es lo que verá:



Cuando no se define su propio RowDefinition y ColumnDefinition objetos, la Cuadrícula los genera automáticamente a medida que se añaden vistas a la Niños colección. Sin embargo, el valor predeterminado

RowDefinition y ColumnDefinition es “*” (asterisco), lo que significa que las cuatro filas ahora igualmente divide la pantalla en cuartos, y cada célula es una octava parte del total Cuadrícula.

Aquí hay otro experimento. restaurar la RowDefinitions y ColumnDefinitions secciones y establecer el HorizontalOptions y VerticalOptions propiedades en el Cuadrícula sí a Centrar. Por defecto estas dos propiedades son Llenar, lo que significa que la Cuadrícula llena su contenedor. Esto es lo que sucede con Centrar:



La tercera fila es todavía el doble de la altura de la fila inferior, pero ahora la altura de la fila inferior se basa en el valor por defecto HeightRequest de BoxView, que es 40.

Verás un efecto similar cuando se pone una Cuadrícula en un StackLayout. También se puede poner una StackLayout en un Cuadrícula celular, u otra Cuadrícula en un Cuadrícula celular, pero no se deje llevar con esta técnica: Cuanto más profundo nido Cuadrícula y otros diseños, más los diseños anidados tendrán un impacto en el rendimiento.

La cuadrícula de código

También es posible definir una Cuadrícula enteramente en código, pero por lo general sin la claridad o el orden de la definición XAML. Los **GridCodeDemo** programa demuestra el enfoque de código mediante la reproducción de la disposición de **SimpleGridDemo**.

Para especificar la altura de una RowDefinition y la anchura de la ColumnDefinition, utiliza valores de la GridLength estructura, a menudo en combinación con el GridUnitType enumeración. Las definiciones de fila hacia la parte superior de la **GridCodeDemoPage** clase demostrar las variaciones de

GridLength. Las definiciones de columna no se incluyen debido a que son los mismos que los generados por defecto:

```
clase pública GridCodeDemoPage : Página de contenido
{
    público GridCodeDemoPage ()
    {
        Cuadrícula rejilla = nuevo Cuadrícula
        {
            RowDefinitions =
            {
                nuevo RowDefinition {Altura = GridLength .auto},
                nuevo RowDefinition {Altura = nuevo GridLength (100)},
                nuevo RowDefinition {Altura = nuevo GridLength (2, GridUnitType .Star)},
                nuevo RowDefinition {Altura = nuevo GridLength (1, GridUnitType .Star)}
            }
        };
        // primera etiqueta (fila 0 y la columna 0).
        grid.Children.Add (nuevo Etiqueta
        {
            text = "Red Demo",
            Tamaño de Letra = Dispositivo .GetNamedSize ( NamedSize .Grande, tipo de ( Etiqueta )),
            HorizontalOptions = LayoutOptions .Fin
        });
        // segunda etiqueta.
        grid.Children.Add (nuevo Etiqueta
        {
            text = "Demostración de la red",
            Tamaño de Letra = Dispositivo .GetNamedSize ( NamedSize .Pequeña, tipo de ( Etiqueta )),
            HorizontalOptions = LayoutOptions .Fin,
            VerticalOptions = LayoutOptions .Fin
        },
        1, // izquierda
        0); // parte superior
        // elemento de imagen.
        grid.Children.Add (nuevo Imagen
        {
            BackgroundColor = Color .Gris,
            fuente = Dispositivo .OnPlatform ( "Icono-60.png",
                "Icon.png",
                "Activos / StoreLogo.png" )
        },
        0, // izquierda
        2, // derecho
        1, // parte superior
        2); // fondo
        // Tres elementos BoxView.
        BoxView boxView1 = nuevo BoxView {Color = Color .Verde};
        Cuadrícula .SetRow (boxView1, 2);
        Cuadrícula .SetColumn (boxView1, 0);
```

```

grid.Children.Add (boxView1);

BoxView boxView2 = nuevo BoxView (Color = Color.Rojo);
Cuadrícula .SetRow (boxView2, 2);
Cuadrícula .SetColumn (boxView2, 1);
Cuadrícula .SetRowSpan (boxView2, 2);
grid.Children.Add (boxView2);

BoxView boxView3 = nuevo BoxView
{
    color = Color.Azul,
    Opacidad = 0,5
};
Cuadrícula .SetRow (boxView3, 3);
Cuadrícula .SetColumn (boxView3, 0);
Cuadrícula .SetColumnSpan (boxView3, 2);
grid.Children.Add (boxView3);

Relleno = nuevo Espesor (0, Dispositivo .OnPlatform (20, 0, 0), 0, 0);
Content = rejilla;
}
}

```

El programa muestra varias maneras diferentes de agregar los niños a la Cuadrícula y especificar las células en las que residen. El primero Etiqueta está en la fila 0 y la columna 0, por lo que sólo necesita ser añadido a la Niños colección de la Cuadrícula obtener configuración predeterminada de filas y columnas:

```

grid.Children.Add (nuevo Etiqueta
{
    ...
});

```

los Cuadrícula redefine su Niños colección para ser de tipo `IGridList <Ver>`, que incluye varios adicional Añadir métodos. Uno de estos Añadir métodos le permite especificar la fila y la columna:

```

grid.Children.Add (nuevo Etiqueta
{
    ...
},
1,           // izquierda
0);          // parte superior

```

A medida que los comentarios indican, los argumentos son en realidad el nombre izquierda y parte superior más bien que columna y fila. Estos nombres tienen más sentido cuando se ve la sintaxis para especificar las filas y columnas:

```

grid.Children.Add (nuevo Imagen
{
    ...
},
0,           // izquierda
2,           // derecho
1,           // parte superior
2);          // fondo

```

Lo que esto significa es que el elemento hijo va en la partida en la columna izquierda pero terminando antes derecho en otras palabras, las columnas 0 y 1. ocupa la fila a partir de parte superior pero terminando antes larva del moscardón-tom, que es la fila 1. El derecho argumento siempre debe ser mayor que izquierda, y el fondo argumento debe ser mayor que parte superior. Si no, el Cuadrícula emite una ArgumentOutOfRangeException.

los `IGridList <Ver>` interfaz también define `AddHorizontal` y `AddVertical` métodos para añadir hijos a una sola fila o columna única Cuadrícula. los Cuadrícula se expande en columnas o filas que se hacen estas llamadas, así como la asignación automática `Grid.Column` o `Grid.Row` configuración de los niños. Usted verá un uso para esta instalación en la siguiente sección.

Al añadir hijos a una Cuadrícula en el código, también es posible hacer llamadas explícitas a `Grid.SetRow`, `Grid.SetColumn`, `Grid.SetRowSpan`, y `Grid.SetColumnSpan`. No importa si usted hace estas llamadas antes o después de agregar el niño al Niños colección de la Cuadrícula:

```
BoxView boxView1 = nuevo BoxView {...};
Cuadrícula .SetRow (boxView1, 2);
Cuadrícula .SetColumn (boxView1, 0);
grid.Children.Add (boxView1);

BoxView boxView2 = nuevo BoxView {...};
Cuadrícula .SetRow (boxView2, 2);
Cuadrícula .SetColumn (boxView2, 1);
Cuadrícula .SetRowSpan (boxView2, 2);
grid.Children.Add (boxView2);

BoxView boxView3 = nuevo BoxView
{
    ...
};
Cuadrícula .SetRow (boxView3, 3);
Cuadrícula .SetColumn (boxView3, 0);
Cuadrícula .SetColumnSpan (boxView3, 2);
grid.Children.Add (boxView3);
```

El gráfico de cuadrícula bar

los `AddVertical` y `AddHorizontal` métodos definidos por la Niños colección de la Cuadrícula

tener la capacidad de añadir toda una colección de puntos de vista a la Cuadrícula en una sola toma. Por defecto, las nuevas filas o columnas reciben una altura o anchura de "*" (asterisco), por lo que la resultante Cuadrícula consta de varias filas o columnas, cada una con el mismo tamaño.

Vamos a usar la `AddHorizontal` método para hacer un poco de gráfico de barras que consiste en 50 `BoxView` elementos con alturas aleatorias. El archivo XAML para el `GridBarChart` programa define una `AbsoluteLayout` fuera es decir los padres a la vez una Cuadrícula y una Marco. Esta Marco sirve como una superposición para mostrar información sobre una barra particular en el gráfico de barras. Tiene su Opacidad establece en 0, por lo que es un principio invisible:

```
< Página de contenido xmlns = "http://xamarin.com/schemas/2014/forms"
    xmlns: x = "http://schemas.microsoft.com/winfx/2009/xaml"
    x: Class = "GridBarChart.GridBarChartPage" >
```

```

< AbsoluteLayout >

    <!-- Rejilla que ocupa toda la página. -->
    < Cuadricula x: Nombre = " cuadricula "
        ColumnSpacing = " 1 "
        AbsoluteLayout.LayoutBounds = " 0, 0, 1, 1 "
        AbsoluteLayout.LayoutFlags = " Todas " />

    <!-- Superposición en el centro de la pantalla. -->
    < Marco x: Nombre = " cubrir "
        OutlineColor = " Acento "
        Color de fondo = "# 404040 "
        Opacidad = " 0 "
        AbsoluteLayout.LayoutBounds = " 0, 0, 1, 1, AutoSize, AutoSize "
        AbsoluteLayout.LayoutFlags = " PositionProportional " />

    < Etiqueta x: Nombre = " etiqueta "
        Color de texto = " Blanco "
        Tamaño de fuente = " Grande " />
    </ Marco >
</ AbsoluteLayout >
</ Pagina de contenido >

```

El archivo de código subyacente crea 50 BoxView elementos con un azar HeightRequest propiedad entre 0 y 300. Además, el styleId propiedad de cada BoxView se le asigna una cadena que consta de consonantes y vocales para parecerse a un nombre al azar alternados (tal vez de alguien de otro planeta). Todos estos BoxView elementos se acumulan en un genérico Lista recogida y después se añadió a la Cuadrícula. Ese trabajo es la mayor parte del código en el constructor:

```

público clase parcial GridBarChartPage : Pagina de contenido
{
    const int COUNT = 50;
    Aleatorio = aleatorios nuevo Aleatorio ();

    público GridBarChartPage ()
    {
        InitializeComponent ();

        Lista < Ver > = vistas nuevo Lista < Ver > ();
        TapGestureRecognizer tapGesture = nuevo TapGestureRecognizer ();
        tapGesture.Tapped += OnBoxViewTapped;

        // Crear elementos BoxView y añadir a la lista.
        para ( En t i = 0; i < COUNT; i++)
        {
            BoxView boxView = nuevo BoxView
            {
                color = Color.Acento,
                HeightRequest = 300 * random.NextDouble (),
                VerticalOptions = LayoutOptions.Fin,
                StyleId = RandomNameGenerator ()
            };
            vistas.Add (boxView);
        }
    }
}

```

```

        boxView.GestureRecognizers.Add (tapGesture);
        views.Add (boxView);
    }

    // Añadir toda la lista de elementos BoxView a la cuadrícula.
    grid.Children.AddHorizontal (vistas);

    // Iniciar un temporizador en la velocidad de fotogramas.
    Dispositivo .StartTimer ( Espacio de tiempo .FromMilliseconds (15), OnTimerTick);
}

// Las matrices para Random Name Generator.

cuerda [] = {Vocales "un", "mí", "yo", "O", "U", "e", "E", "es decir", "UNED", "Oo"};
cuerda [] Consonantes = { "segundo", "do", "re", "F", "gramo", "H", "J", "K", "L", "metro",
    "norte", "peg", "Q", "T", "S", "T", "V", "W", "X", "Z" };

cuerda RandomNameGenerator ()
{
    En t numPieces = 1 + 2 * random.Next (1, 4);
    StringBuilder name = nuevo StringBuilder ();

    para ( En t i = 0; i <numPieces; i++)
    {
        name.Append (i% 2 == 0?
            consonantes [random.Next (consonants.Length)]:
            vocales [random.Next (vocales.Length)]);
    }
    Nombre [0] = Carbonizarse .ToUpper (nombre [0]);
    regreso name.ToString ();
}

// Establecer el texto de la superposición de etiquetas y hacerla visible.
void OnBoxViewTapped ( objeto remitente, EventArgs args)
{
    BoxView boxView = ( BoxView )remitente;
    label.text = Cuerda .Formato ("El individuo conocido como {0}" +
        "Tiene una altura de {1} centímetros.",

        boxView.StyleId, ( En t ) BoxView.HeightRequest);
    overlay.Opacity = 1;
}

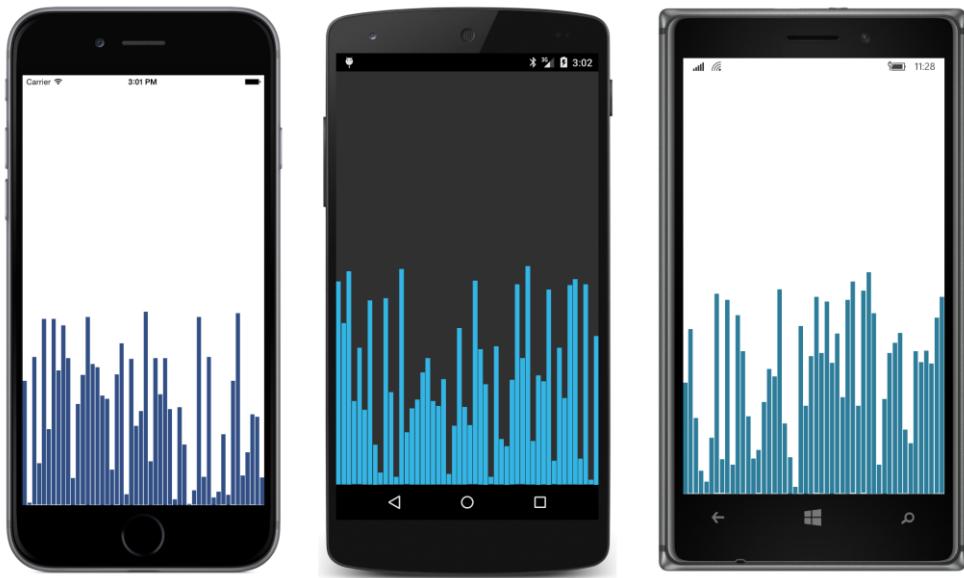
// visibilidad Disminución de superposición.
bool OnTimerTick ()
{
    overlay.Opacity = Mates .MAX (0, overlay.Opacity - 0,0025);
    return true ;
}
}
}

```

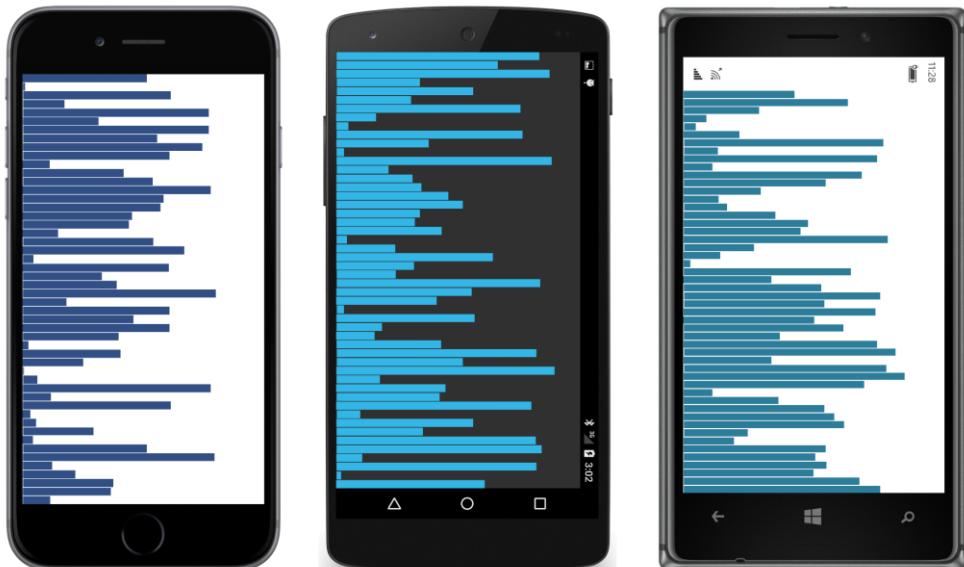
los AddHorizontal método de la Niños colección añade el múltiple BoxView elementos a la Cuadrícula y les da secuencial Grid.Column ajustes.

Cada columna de forma predeterminada tiene una anchura de "*" (estrellas), por lo que la anchura de cada BoxView es la misma mientras que la altura se rige por la HeightRequest

ajustes. Los Espaciado valor de 1 conjunto a la Cuadrícula en el archivo XAML proporciona un poco la separación entre las barras de la gráfica de barras:

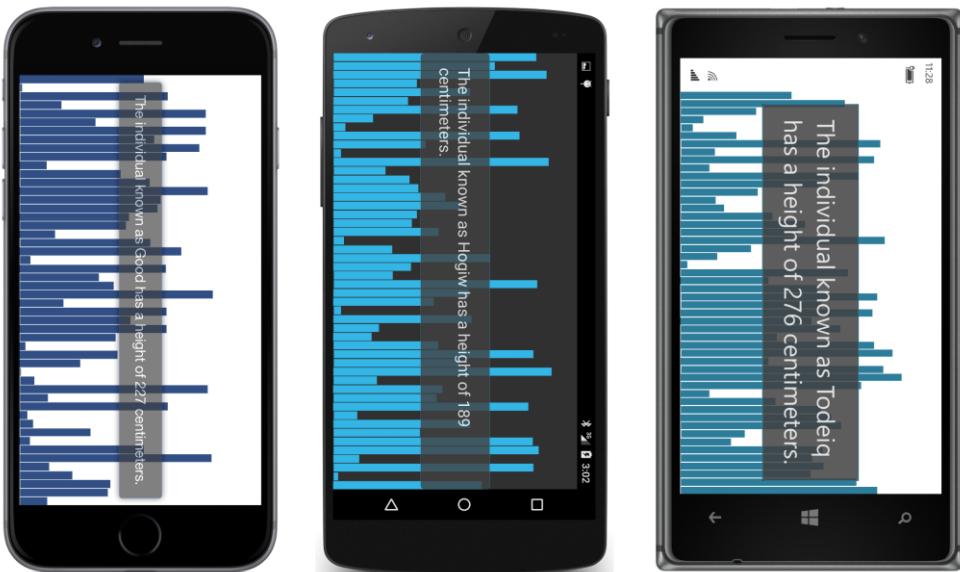


Las barras son más claras cuando se enciende el teléfono de lado para darles más ancho:



Este programa tiene otra característica: Cuando toca en uno de los bares, la superposición de la información se hace visible y muestra información sobre esa barra, específicamente, el nombre del visitante interplanetaria sangrado del

styleId y la altura de la barra. Sin embargo, un temporizador establecido en el constructor disminuye continuamente la Opacidad valor en la superposición, por lo que esta información se desvanece poco a poco a la vista:



Incluso sin un sistema de gráficos nativo, Xamarin.Forms es capaz de mostrar algo que se parece mucho al igual que los gráficos.

La alineación en la cuadrícula

UN Cuadrícula con una fila Altura propiedad de Auto limita la altura de los elementos de esa fila de la misma manera como una vertical de StackLayout. Del mismo modo, una columna con una Anchura de Auto funciona como una horizontal StackLayout.

Como hemos visto anteriormente en este capítulo, se puede establecer el HorizontalOptions y VerticalOptions propiedades de los hijos de la Cuadricula a la posición de ellos dentro de la célula. Aquí hay un programa llamado GridAlignment que crea una Cuadricula con nueve celdas de igual tamaño y luego pone seis Etiqueta todos los elementos en la celda central, pero con diferentes parámetros de alineación:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " GridAlignment.GridAlignmentPage " >

< Cuadricula >
    < Grid.RowDefinitions >
        < RowDefinition Altura = " * " />
        < RowDefinition Altura = " * " />
        < RowDefinition Altura = " * " />
    </ Grid.RowDefinitions >
```

```
< Grid.ColumnDefinitions >
    < ColumnDefinition Anchura = " * " />
    < ColumnDefinition Anchura = " * " />
    < ColumnDefinition Anchura = " * " />
</ Grid.ColumnDefinitions >

< Etiqueta Texto = " Arriba a la izquierda "
    Grid.Row = " 1 " Grid.Column = " 1 "
    VerticalOptions = " comienzo "
    HorizontalOptions = " comienzo " />

< Etiqueta Texto = " Superior derecha "
    Grid.Row = " 1 " Grid.Column = " 1 "
    VerticalOptions = " comienzo "
    HorizontalOptions = " Fin " />

< Etiqueta Texto = " izquierda centro "
    Grid.Row = " 1 " Grid.Column = " 1 "
    VerticalOptions = " Centrar "
    HorizontalOptions = " comienzo " />

< Etiqueta Texto = " centro de derecho "
    Grid.Row = " 1 " Grid.Column = " 1 "
    VerticalOptions = " Centrar "
    HorizontalOptions = " Fin " />

< Etiqueta Texto = " Abajo a la izquierda "
    Grid.Row = " 1 " Grid.Column = " 1 "
    VerticalOptions = " Fin "
    HorizontalOptions = " comienzo " />

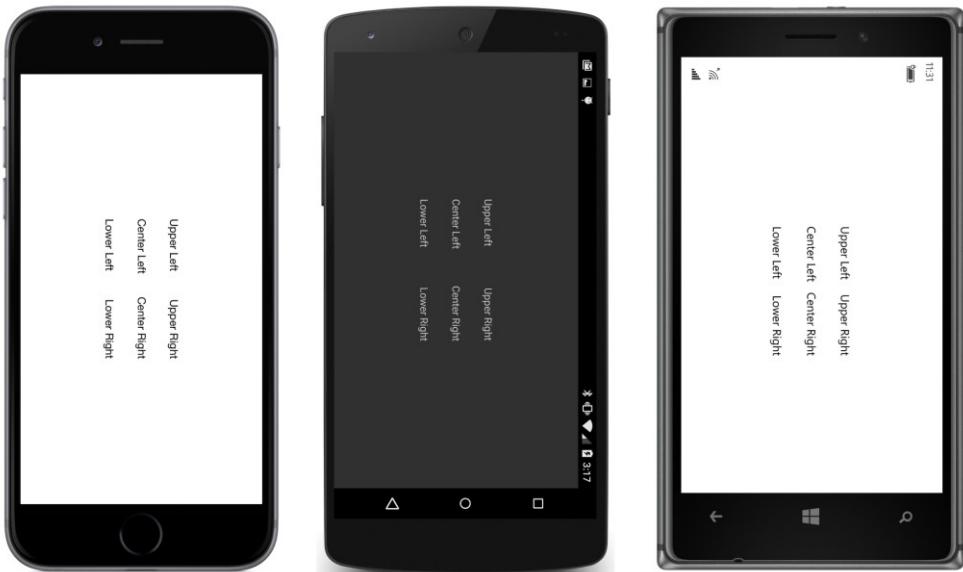
< Etiqueta Texto = " Inferior derecha "
    Grid.Row = " 1 " Grid.Column = " 1 "
    VerticalOptions = " Fin "
    HorizontalOptions = " Fin " />

</ Cuadricula >
</ Pagina de contenido >
```

Como se puede ver, la parte del texto se superpone:



Pero si a su vez los lados del teléfono, cambiar el tamaño de las células y el texto no se superponen:



Aunque se puede utilizar `HorizontalOptions` y `VerticalOptions` en los niños de una Cuadrícula para definir la alineación del niño, no se puede utilizar el `expande bandera`. Estrictamente hablando, en realidad se *puede* utilizar el `Expander`, pero no tiene ningún efecto en los niños de una Cuadrícula. Los `expande bandera` sólo afecta a los niños de una `StackLayout`.

A menudo se ha visto programas que utilizan el `expande bandera` para los niños de una `StackLayout` Para proveer

espacio adicional para rodear los elementos dentro de la disposición. Por ejemplo, si dos Etiqueta hijos de una Apilar-Diseño ambos tienen su VerticalOptions propiedades ajustado a CenterAndExpand, entonces todo el espacio extra se divide en partes iguales entre las dos ranuras en el StackLayout asignados para estos niños.

en un Cuadrícula, se puede realizar trucos de diseño similares mediante el uso de células de tamaño con el "*" especificación (estrella), junto con HorizontalOptions y VerticalOptions configuración de los niños. Incluso puede crear filas o columnas vacías vacías simplemente para espaciar los propósitos.

los SpacingButtons programa igualmente espacios de tres botones verticales y horizontales tres botones. Los tres primeros botones ocupan una fila de tres Cuadrícula que ocupa gran parte de la página, y los tres botones horizontales se encuentran en un período de tres columnas Cuadrícula hacia abajo en la parte inferior de la página. Las dos rejillas están en una StackLayout:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " SpacingButtons.SpacingButtonsPage " >

    < StackLayout >
        < Cuadrícula VerticalOptions = " FillAndExpand " >
            < Grid.RowDefinitions >
                < RowDefinition Altura = " * " />
                < RowDefinition Altura = " * " />
                < RowDefinition Altura = " * " />
            </ Grid.RowDefinitions >

            < Botón Texto = " Botón 1 " 
                Grid.Row = " 0 "
                VerticalOptions = " Centrar "
                HorizontalOptions = " Centrar " />

            < Botón Texto = " botón 2 "
                Grid.Row = " 1 "
                VerticalOptions = " Centrar "
                HorizontalOptions = " Centrar " />

            < Botón Texto = " botón 3 "
                Grid.Row = " 2 "
                VerticalOptions = " Centrar "
                HorizontalOptions = " Centrar " />

        </ Cuadrícula >

        < Cuadrícula >
            < Grid.ColumnDefinitions >
                < ColumnDefinition Anchura = " * " />
                < ColumnDefinition Anchura = " * " />
                < ColumnDefinition Anchura = " * " />
            </ Grid.ColumnDefinitions >

            < Botón Texto = " botón 4 "
                Grid.Column = " 0 "
                HorizontalOptions = " Centrar " />

            < Botón Texto = " botón 5 " >
```

```

Grid.Column = "1"
HorizontalOptions = "Centrar" />

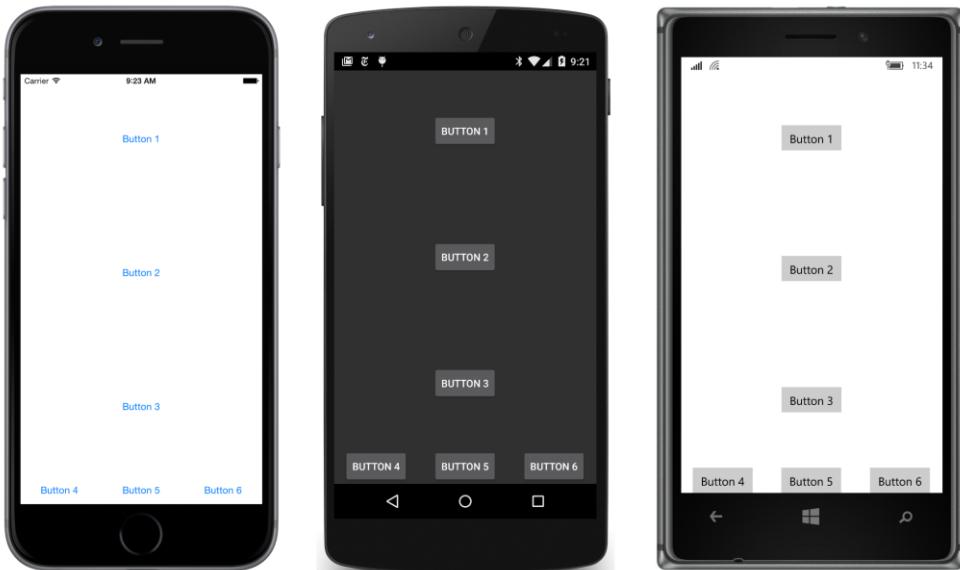
<Botón Texto = "botón 6"
    Grid.Column = "2"
    HorizontalOptions = "Centrar" />

</Cuadrícula>
</StackLayout>
</Página de contenido>

```

El segundo Cuadrícula tiene un defecto VerticalOptions valor de Llenar, mientras que la primera Cuadrícula tiene una configuración explícita para VerticalOptions a FillAndExpand. Esto significa que la primera Cuadrícula ocupará toda la superficie de la pantalla no ocupada por el segundo Cuadrícula. El tres RowDefinition objetos de la primera

Cuadrícula dividir esa área en tres partes. Dentro de cada célula, el Botón es horizontal y centrado verticalmente:



El segundo Cuadrícula divide su área en tres columnas igualmente espaciados, y cada Botón se centra horizontalmente dentro de esa área.

Aunque el expand Bandera de LayoutOptions puede ayudar en objetos visuales igualmente espacio dentro de una StackLayout, la técnica se rompe cuando los objetos visuales no son de un tamaño uniforme. las Exmarcas de fáctica opción asigna el espacio sobrante en partes iguales entre todas las ranuras de la StackLayout, pero el tamaño total de cada ranura depende del tamaño de los objetos visuales individuales. los Cuadrícula, sin embargo, asigna espacio igualmente a las células, y luego los objetos visuales están alineados dentro de ese espacio.

divisores celulares y fronteras

los Cuadrícula no tiene un built-in separadores celulares o fronteras. Pero si desea alguna, se les puede añadir a sí mismo. los **GridCellDividers** programa define una **GridLength** valor en su recursos diccionario

llamado `dividerThickness`. Esto se utiliza para la altura y la anchura de cada otra fila y columna en la Cuadrícula. La idea aquí es que estas filas y columnas son de los divisores, mientras que las otras filas y columnas son para el contenido normal:

```
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns:x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " GridCellDividers.GridCellDividersPage " >

    < ContentPage.Padding >
        < OnPlatform x: TypeArguments = " Espesor "
            iOS = " 0, 20, 0, 0 "
            Androide = " 0 "
            WinPhone = " 0 " />
    </ ContentPage.Padding >

    < Cuadrícula >
        < Grid.Resources >
            < ResourceDictionary >
                < GridLength x: Key = " dividerThickness " > 2 </ GridLength >

                < Estilo Tipo de objetivo = " BoxView " >
                    < Setter Propiedad = " Color " Valor = " Acento " />
                </ Estilo >

                < Estilo Tipo de objetivo = " Etiqueta " >
                    < Setter Propiedad = " HorizontalOptions " Valor = " Centrar " />
                    < Setter Propiedad = " VerticalOptions " Valor = " Centrar " />
                </ Estilo >
            </ ResourceDictionary >
        </ Grid.Resources >

        < Grid.RowDefinitions >
            < RowDefinition Altura = " 0 " StaticResource dividerThickness " />
            < RowDefinition Altura = " * " />
            < RowDefinition Altura = " 0 " StaticResource dividerThickness " />
            < RowDefinition Altura = " * " />
            < RowDefinition Altura = " 0 " StaticResource dividerThickness " />
            < RowDefinition Altura = " * " />
            < RowDefinition Altura = " 0 " StaticResource dividerThickness " />
        </ Grid.RowDefinitions >

        < Grid.ColumnDefinitions >
            < ColumnDefinition Anchura = " 0 " StaticResource dividerThickness " />
            < ColumnDefinition Anchura = " * " />
            < ColumnDefinition Anchura = " 0 " StaticResource dividerThickness " />
            < ColumnDefinition Anchura = " * " />
            < ColumnDefinition Anchura = " 0 " StaticResource dividerThickness " />
            < ColumnDefinition Anchura = " * " />
            < ColumnDefinition Anchura = " 0 " StaticResource dividerThickness " />
        </ Grid.ColumnDefinitions >

        < BoxView Grid.Row = " 0 " Grid.Column = " 0 " Grid.ColumnSpan = " 7 " />
        < BoxView Grid.Row = " 2 " Grid.Column = " 0 " Grid.ColumnSpan = " 7 " />
        < BoxView Grid.Row = " 4 " Grid.Column = " 0 " Grid.ColumnSpan = " 7 " />
```

```

< BoxView Grid.Row = "6" Grid.Column = "0" Grid.ColumnSpan = "7" />

< BoxView Grid.Row = "0" Grid.Column = "0" Grid.RowSpan = "7" />
< BoxView Grid.Row = "0" Grid.Column = "2" Grid.RowSpan = "7" />
< BoxView Grid.Row = "0" Grid.Column = "4" Grid.RowSpan = "7" />
< BoxView Grid.Row = "0" Grid.Column = "6" Grid.RowSpan = "7" />

< Etiqueta Texto = "Cuadrícula"
    Grid.Row = "1" Grid.Column = "1" />

< Etiqueta Texto = "Celda"
    Grid.Row = "3" Grid.Column = "3" />

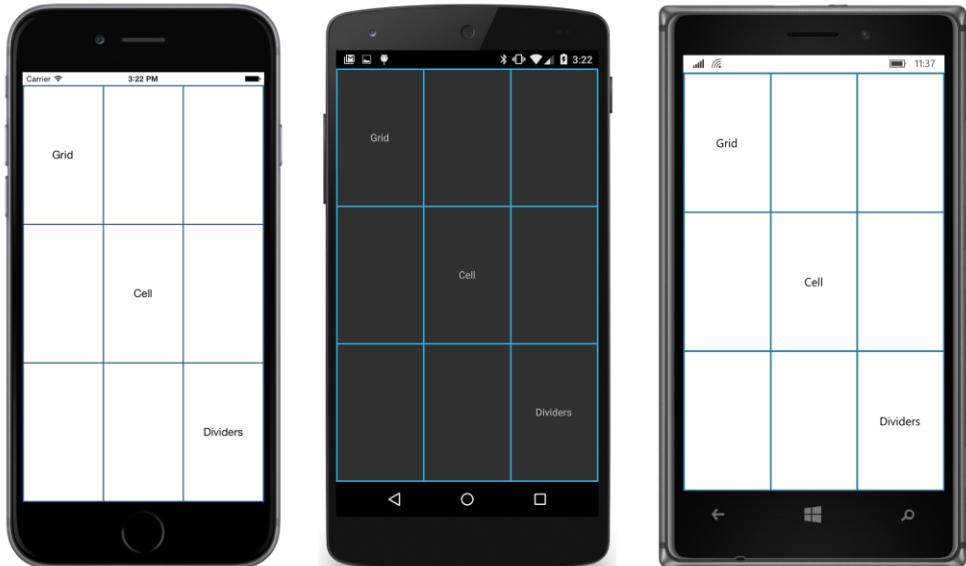
< Etiqueta Texto = "divisores"
    Grid.Row = "5" Grid.Column = "5" />

</ Cuadrícula >
</ Pagina de contenido >

```

Cada fila y columna de los divisores es ocupada por una BoxView coloreada con el Acento el color de un estilo implícita. Para los divisores horizontales, la altura se establece por el RowDefinition y la anchura se rige por la Grid.ColumnSpan adjunta propiedad enlazable; un enfoque similar se aplica para los divisores verticales.

los Cuadrícula También contiene tres Etiqueta Elementos sólo para demostrar cómo regular de contenido encaja con estos divisores:



No es necesario asignar filas y columnas enteras a estos divisores. Tenga en cuenta que los objetos visuales pueden compartir las células, por lo que es posible añadir una BoxView (o dos o tres o cuatro) a una célula y configurar las opciones horizontal y vertical para que abraza a la pared de la célula y se asemeja a una frontera.

He aquí un programa similar, llamado **GridCellBorders**, que muestra el contenido en los mismos tres células como **GridCellDividers**, pero esos tres células también están adornados con bordes.

los recursos El diccionario contiene no menos de siete estilos que se dirigen BoxView! El estilo de base establece el color, dos estilos más Establecer el HeightRequest y WidthRequest para los bordes horizontales y verticales, y luego establecen cuatro estilos más la VerticalOptions a comienzo o Fin para los bordes superior e inferior y HorizontalOptions a comienzo y Fin para los bordes izquierdo y derecho. los BorderThickness entrada de diccionario es una doble porque se usa para establecer WidthRequest y HeightRequest propiedades de la BoxView elementos:

```
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns:x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " GridCellBorders.GridCellBordersPage ">

    < ContentPage.Padding >
        < OnPlatform x: TypeArguments = " Espesor "
            iOS = " 10, 20, 10, 10 "
            Androide = " 10 "
            WinPhone = " 10 " />
    </ ContentPage.Padding >

    < Cuadrícula >
        < Grid.Resources >
            < ResourceDictionary >
                < x: Doble x: Key = " BorderThickness " > 1 </ x: Doble >

                    < Estilo x: Key = " baseBorderStyle " Tipo de objetivo = " BoxView " >
                        < Setter Propiedad = " Color " Valor = " Acento " />
                    </ Estilo >

                    < Estilo x: Key = " horzBorderStyle " Tipo de objetivo = " BoxView " >
                        Residencia en = " () StaticResource baseBorderStyle " >
                        < Setter Propiedad = " HeightRequest " Valor = " () StaticResource BorderThickness " />
                    </ Estilo >

                    < Estilo x: Key = " topBorderStyle " Tipo de objetivo = " BoxView " >
                        Residencia en = " () StaticResource horzBorderStyle " >
                        < Setter Propiedad = " VerticalOptions " Valor = " comienzo " />
                    </ Estilo >

                    < Estilo x: Key = " bottomBorderStyle " Tipo de objetivo = " BoxView " >
                        Residencia en = " () StaticResource horzBorderStyle " >
                        < Setter Propiedad = " VerticalOptions " Valor = " Fin " />
                    </ Estilo >

                    < Estilo x: Key = " vertBorderStyle " Tipo de objetivo = " BoxView " >
                        Residencia en = " () StaticResource baseBorderStyle " >
                        < Setter Propiedad = " WidthRequest " Valor = " () StaticResource BorderThickness " />
                    </ Estilo >

                    < Estilo x: Key = " leftBorderStyle " Tipo de objetivo = " BoxView " >
                        Residencia en = " () StaticResource vertBorderStyle " >
                    </ Estilo >
            </ ResourceDictionary >
        </ Grid.Resources >
    </ Cuadrícula >
</ Pagina de contenido >
```

```
< Setter Propiedad = "HorizontalOptions" Valor = "comienzo" />
</ Estilo >

< Estilo x: Key = "rightBorderStyle" Tipo de objetivo = "BoxView" 
Residencia en = " {} StaticResource vertBorderStyle" >
< Setter Propiedad = "HorizontalOptions" Valor = "Fin" />
</ Estilo >

< Estilo Tipo de objetivo = "Etiqueta" >
< Setter Propiedad = "HorizontalOptions" Valor = "Centrar" />
< Setter Propiedad = "VerticalOptions" Valor = "Centrar" />
</ Estilo >
</ ResourceDictionary >
</ Grid.Resources >

< Grid.RowDefinitions >
< RowDefinition Altura = "*" />
< RowDefinition Altura = "*" />
< RowDefinition Altura = "*" />
</ Grid.RowDefinitions >

< Grid.ColumnDefinitions >
< ColumnDefinition Anchura = "*" />
< ColumnDefinition Anchura = "*" />
< ColumnDefinition Anchura = "*" />
</ Grid.ColumnDefinitions >

< Etiqueta Texto = "Cuadrícula" 
Grid.Row = "0" Grid.Column = "0" />

< BoxView Estilo = " {} StaticResource topBorderStyle" 
Grid.Row = "0" Grid.Column = "0" />

< BoxView Estilo = " {} StaticResource bottomBorderStyle" 
Grid.Row = "0" Grid.Column = "0" />

< BoxView Estilo = " {} StaticResource leftBorderStyle" 
Grid.Row = "0" Grid.Column = "0" />

< BoxView Estilo = " {} StaticResource rightBorderStyle" 
Grid.Row = "0" Grid.Column = "0" />

< Cuadrícula Grid.Row = "1" Grid.Column = "1" >
< Etiqueta Texto = "Celda" />
< BoxView Estilo = " {} StaticResource topBorderStyle" />
< BoxView Estilo = " {} StaticResource bottomBorderStyle" />
< BoxView Estilo = " {} StaticResource leftBorderStyle" />
< BoxView Estilo = " {} StaticResource rightBorderStyle" />
</ Cuadrícula >

< Cuadrícula Grid.Row = "2" Grid.Column = "2" >
< Etiqueta Texto = "fronteras" />
< BoxView Estilo = " {} StaticResource topBorderStyle" />
< BoxView Estilo = " {} StaticResource bottomBorderStyle" />
```

```

< BoxView Estilo = " {> StaticResource leftBorderStyle " />
< BoxView Estilo = " {> StaticResource rightBorderStyle " />
</ Cuadrícula >
</ Cuadrícula >
</ Pagina de contenido >

```

En la celda de la esquina superior izquierda, el Etiqueta y cuatro BoxView elementos, cada uno recibe su Grid.Row y Grid.Column atributos establecidos a 0. Sin embargo, para el medio Cuadrícula y la parte inferior derecha Cuadrícula, un enfoque bastante más fácil es tomada: Otra Cuadrícula con una sola célula ocupa la célula, y que una sola célula Cuadrícula contiene el Etiqueta y cuatro BoxView elementos. Los resultados de la simplicidad de la configuración Grid.Row y Grid.Column sólo en el de una sola célula Cuadrícula:

```

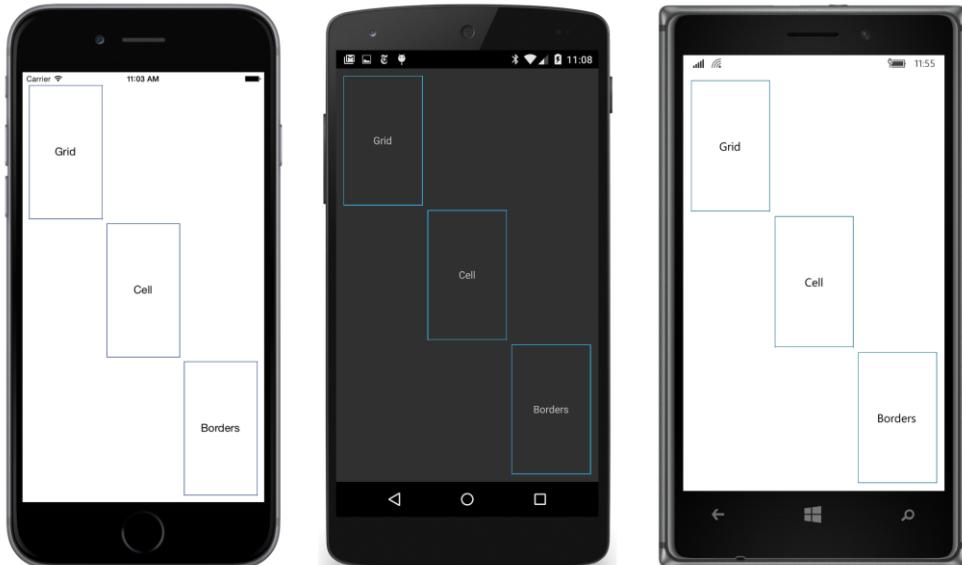
< Cuadrícula Grid.Row = " 1 " Grid.Column = " 1 " >
< Etiqueta Texto = " Celda " />
< BoxView Estilo = " {> StaticResource topBorderStyle " />
< BoxView Estilo = " {> StaticResource bottomBorderStyle " />
< BoxView Estilo = " {> StaticResource leftBorderStyle " />
< BoxView Estilo = " {> StaticResource rightBorderStyle " />
</ Cuadrícula >

```

Al anidar una Cuadrícula dentro de otro Cuadrícula, el uso de la Grid.Row y Grid.Column atributos pueden ser confusos. Esta sola célula Cuadrícula ocupa la segunda fila y la segunda columna de su padre, que es el Cuadrícula que ocupa toda la página.

Además, tenga en cuenta que cuando una Cuadrícula se pone a sí misma, se ve sólo en el Grid.Row y Grid.Column la configuración de sus hijos, y nunca sus nietos u otros descendientes en el árbol visual.

Aquí está el resultado:



Podría ser un poco desconcertante que las esquinas de las fronteras no cumplen, pero eso es debido a la fila y la columna de separación por defecto de la Cuadrícula. Selecciona el Distancia entre filas y ColumnSpacing atribuye a 0, y las esquinas se reunirán aunque las líneas seguirán parecer un tanto discontinua debido a que las fronteras están en diferentes células. Si esto es inaceptable, utilice la técnica mostrada en **GridCellDividers**.

Si desea que todas las filas y columnas muestran con separadores como en **GridCellDividers**, Otra técnica es establecer la Color de fondo propiedad de la Cuadrícula y el uso de la Distancia entre filas y ColumnSpacing

propiedades que permiten que el color vistazo a través de los espacios entre las células. Pero todas las células deben contener contenido que tiene un fondo opaco para que esta técnica sea visualmente convincente.

Casi ejemplos de cuadrícula de la vida real

Ahora estamos listos para volver a escribir la XamlKeypad programa desde el capítulo 8 para utilizar una Cuadrícula. La nueva versión se llama **KeypadGrid**. El uso de una Cuadrícula no sólo obliga al Botón elementos que componen el teclado sean todos del mismo tamaño, sino que también permite a los componentes del teclado para abarcar las células.

los Cuadrícula que conforma el teclado está centrada en la página con HorizontalOptions y verticalOptions ajustes. Cuenta con cinco filas y tres columnas, pero el RowDefinitions y ColumnDefinitions colecciones no tienen que ser construidos de manera explícita, porque cada célula tiene una altura "*" (asterisco) y el ancho.

Por otra parte, la totalidad Cuadrícula se da una plataforma específica WidthRequest y HeightRequest, donde la anchura es tres quintas partes de la altura. (La diferencia para Windows Phone se basa en el tamaño algo mayor de la Grande tamaño de fuente utilizado para la Botón.) Esto hace que cada célula en el Cuadrícula Será cuadrado:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " KeypadGrid.KeypadGridPage " >

    < Cuadrícula Distancia entre filas = " 2 "
        ColumnSpacing = " 2 "
        VerticalOptions = " Centrar "
        HorizontalOptions = " Centrar " >
        < Grid.WidthRequest >
            < OnPlatform x: TypeArguments = " x: Doble "
                iOS = " 180 "
                Androide = " 180 "
                WinPhone = " 240 " />
        </ Grid.WidthRequest >

        < Grid.HeightRequest >
            < OnPlatform x: TypeArguments = " x: Doble "
                iOS = " 300 "
                Androide = " 300 "
                WinPhone = " 400 " />
        </ Grid.HeightRequest >

    < Grid.Resources >
```

```
< ResourceDictionary >
    < Estilo Tipo de objetivo = " Botón " >
        < Setter Propiedad = " Tamaño de fuente " Valor = " Grande " />
        < Setter Propiedad = " Ancho del borde " Valor = " 1 " />
    </ Estilo >
</ ResourceDictionary >
</ Grid.Resources >

< Etiqueta x: Nombre = " displayLabel "
    Grid.Row = " 0 " Grid.Column = " 0 " Grid.ColumnSpan = " 2 "
    Tamaño de fuente = " Grande "
    LineBreakMode = " HeadTruncation "
    VerticalOptions = " Centrar "
    HorizontalTextAlignment = " Fin " />

< Botón x: Nombre = " backspaceButton "
    Texto = " & # X21E6; "
    Grid.Row = " 0 " Grid.Column = " 2 "
    Está habilitado = " Falso "
    hecho clic = " OnBackspaceButtonClicked " />

< Botón Texto = " 7 " styleid = " 7 "
    Grid.Row = " 1 " Grid.Column = " 0 "
    hecho clic = " OnDigitButtonClicked " />

< Botón Texto = " 8 " styleid = " 8 "
    Grid.Row = " 1 " Grid.Column = " 1 "
    hecho clic = " OnDigitButtonClicked " />

< Botón Texto = " 9 " styleid = " 9 "
    Grid.Row = " 1 " Grid.Column = " 2 "
    hecho clic = " OnDigitButtonClicked " />

< Botón Texto = " 4 " styleid = " 4 "
    Grid.Row = " 2 " Grid.Column = " 0 "
    hecho clic = " OnDigitButtonClicked " />

< Botón Texto = " 5 " styleid = " 5 "
    Grid.Row = " 2 " Grid.Column = " 1 "
    hecho clic = " OnDigitButtonClicked " />

< Botón Texto = " 6 " styleid = " 6 "
    Grid.Row = " 2 " Grid.Column = " 2 "
    hecho clic = " OnDigitButtonClicked " />

< Botón Texto = " 1 " styleid = " 1 "
    Grid.Row = " 3 " Grid.Column = " 0 "
    hecho clic = " OnDigitButtonClicked " />

< Botón Texto = " 2 " styleid = " 2 "
    Grid.Row = " 3 " Grid.Column = " 1 "
    hecho clic = " OnDigitButtonClicked " />

< Botón Texto = " 3 " styleid = " 3 "
```

```

Grid.Row = "3" Grid.Column = "2"
hecho clic = "OnDigitButtonClicked" />

< Botón Texto = "0" styleid = "0"
    Grid.Row = "4" Grid.Column = "0" Grid.ColumnSpan = "2"
    hecho clic = "OnDigitButtonClicked" />

< Botón Texto = "." styleid = "."
    Grid.Row = "4" Grid.Column = "2"
    hecho clic = "OnDigitButtonClicked" />

</ Cuadrícula >
</ Página de contenido >

```

los Etiqueta y el botón de retroceso ocupan la primera fila, pero el Etiqueta abarca dos columnas y el botón de retroceso se encuentra en la tercera columna. Del mismo modo, la fila inferior de la Cuadrícula contiene el botón de cero y el botón de punto decimal, pero el botón de cero se extiende por dos columnas como es típico en los teclados de ordenador.

El archivo de código subyacente es el mismo que el **XamlKeypad** programa. Además, el programa guarda las entradas cuando el programa se pone a dormir y luego los restaura cuando el programa se pone en marcha de nuevo. Un borde se ha añadido a la Botón en un estilo implícita para que se vea más como un teclado real en iOS:



Como se puede recordar, la **OnDigitButtonClicked** manejador en el archivo de código subyacente utiliza el **styleid** propiedad para anexar un nuevo carácter a la cadena de texto. Pero como se puede ver en el archivo XAML, para cada uno de los botones con este controlador de eventos, la **styleid** se establece en el mismo carácter que la **Texto** propiedad de la Botón. No puede el controlador de eventos utilice en su lugar?

Sí puede. Pero suponga que decide que el punto decimal en el Botón no se presenta muy bien.

Es posible que prefiera utilizar un punto más pesado y más central, tales como \u00B7 (llamado punto medio) o \u22C5 (la matemática operador punto) o incluso \u2022 (la bala). Tal vez le gusta también diferentes estilos de números para estos otros botones, como el conjunto de números que comienzan en cercados \u2460 en el estándar Unicode, o los números romanos que comienzan en \u2160. Puede reemplazar el `Texto` propiedad en el archivo XAML sin tocar el archivo de código subyacente:



los `styleid` es una de las herramientas para mantener el aspecto visual y mecánica de la interfaz de usuario restringido para el marcado y separado de su código. Usted verá más herramientas para estructurar su programa en el siguiente capítulo, que cubre la arquitectura de la aplicación Model-View-ViewModel. Ese capítulo también presenta una variación del programa de teclado convertido en una máquina de sumar.

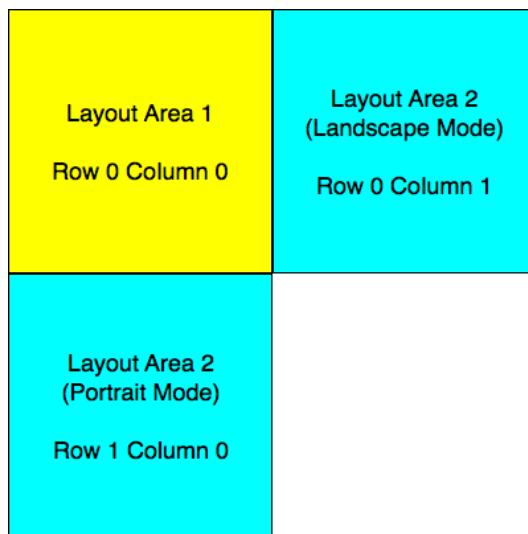
En respuesta a los cambios de orientación

El diseño de la página de una aplicación está generalmente ligada muy de cerca a una relación de factor de forma y aspecto particular. A veces, una aplicación requiere que se use sólo en modo retrato o paisaje. Pero por lo general una aplicación intentará mover las cosas en la pantalla cuando la orientación del teléfono cambia.

UN Cuadrícula puede ayudar a una aplicación acomodarse a los cambios de orientación. los Cuadrícula se puede definir en XAML con ciertas provisiones para ambos modos vertical y horizontal, y luego un poco de código puede hacer los ajustes necesarios dentro de una `SizeChanged` manejador de la página.

Este trabajo es más fácil si se puede dividir toda la disposición de su solicitud en dos grandes áreas que se pueden organizar verticalmente cuando el teléfono está orientado en modo vertical u horizontal para el modo paisaje. Poner cada una de estas áreas en las células separadas de una Cuadrícula. Cuando el teléfono está en el modo vertical, la Cuadrícula tiene dos filas, y cuando está en modo horizontal, que tiene dos columnas. En el diagrama siguiente, la primera área

es siempre en la parte superior o la izquierda. La segunda zona puede ser ya sea en la segunda fila para el modo de retrato o la segunda columna para el modo de paisaje:



Para mantener las cosas razonablemente simple, tendrá que definir el Cuadrícula en XAML con dos filas y dos columnas, pero en modo vertical, la segunda columna tiene una anchura de cero, y en el modo horizontal de la segunda fila tiene una altura cero.

los **GridRgbSliders** programa muestra esta técnica. Es similar a la **RgbSliders** programa desde el capítulo 15, “La interfaz interactiva”, excepto que el diseño utiliza una combinación de una Cuadrícula y una **StackLayout**, y el Etiqueta elementos de visualización los valores actuales de la deslizador elementos mediante el uso de enlaces de datos con un convertidor de valores y un parámetro de convertidor de valores. (Más sobre esto más adelante.) Ajuste de la **Color** propiedad de la **BoxView** basado en los tres deslizador elementos todavía requiere un código debido a que el R, GRAMO, y segundo propiedades de la Color struct no están respaldados por propiedades enlazables, y estas propiedades no se puede cambiar de forma individual todos modos, ya que no tienen público conjunto descriptores de acceso. (Sin embargo, en el siguiente capítulo, en MVVM, verá una forma de eliminar esta lógica en el archivo de código subyacente.)

Como se puede ver en el listado siguiente, el Cuadrícula llamado mainGrid tiene de hecho dos filas y dos columnas. Sin embargo, se inicializan para modo vertical, por lo que la segunda columna tiene una anchura de cero. La fila superior de la Cuadrícula contiene el BoxView, y que ha hecho lo más grande posible con un ajuste “*” (asterisco), mientras que la fila inferior contiene una StackLayout con todos los controles interactivos. Esto se da una altura de Auto:

```
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
  xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
  xmlns: kit de herramientas =
    "clr-espacio de nombres: Xamarin.FormsBook.Toolkit; montaje = Xamarin.FormsBook.Toolkit "
  x: Class = " GridRgbSliders.GridRgbSlidersPage "
  SizeChanged = " OnPageSizeChanged " >

< ContentPage.Padding >
```

```
< OnPlatform x: TypeArguments = " Espesor "
    iOS = " 0, 20, 0, 0 " />

</ ContentPage.Padding >

< ContentPage.Resources >
    < ResourceDictionary >
        < kit de herramientas: DoubleToIntConverter x: Key = " doubleToInt " />

        < Estilo Tipo de objetivo = " Etiqueta " >
            < Setter Propiedad = " HorizontalTextAlignment " Valor = " Centrar " />
        </ Estilo >
    </ ResourceDictionary >
</ ContentPage.Resources >

< Cuadrícula x: Nombre = " mainGrid " >
    <!-- Inicializado para el modo de retrato. -->
    < Grid.RowDefinitions >
        < RowDefinition Altura = " * " />
        < RowDefinition Altura = " Auto " />
    </ Grid.RowDefinitions >

    < Grid.ColumnDefinitions >
        < ColumnDefinition Anchura = " * " />
        < ColumnDefinition Anchura = " 0 " />
    </ Grid.ColumnDefinitions >

    < BoxView x: Nombre = " boxView "
        Grid.Row = " 0 " Grid.Column = " 0 " />

    < StackLayout x: Nombre = " controlPanelStack "
        Grid.Row = " 1 " Grid.Column = " 0 "
        Relleno = " 10, 5 " >

        < StackLayout VerticalOptions = " CenterAndExpand " >
            < deslizador x: Nombre = " redSlider "
                ValueChanged = " OnSliderValueChanged " />

            < Etiqueta Texto = " {Binding Fuente = {x: redSlider Referencia},
                Path = Valor,
                Convertidor = {} StaticResource doubleToInt,
                ConverterParameter = 255,
                StringFormat = 'Rojo = {0: X2}' } " />
        </ StackLayout >

        < StackLayout VerticalOptions = " CenterAndExpand " >
            < deslizador x: Nombre = " greenSlider "
                ValueChanged = " OnSliderValueChanged " />

            < Etiqueta Texto = " {Binding Fuente = {x: greenSlider Referencia},
                Path = Valor,
                Convertidor = {} StaticResource doubleToInt,
                ConverterParameter = 255,
                StringFormat = 'Verde = {0: X2}' } " />
        </ StackLayout >
```

```

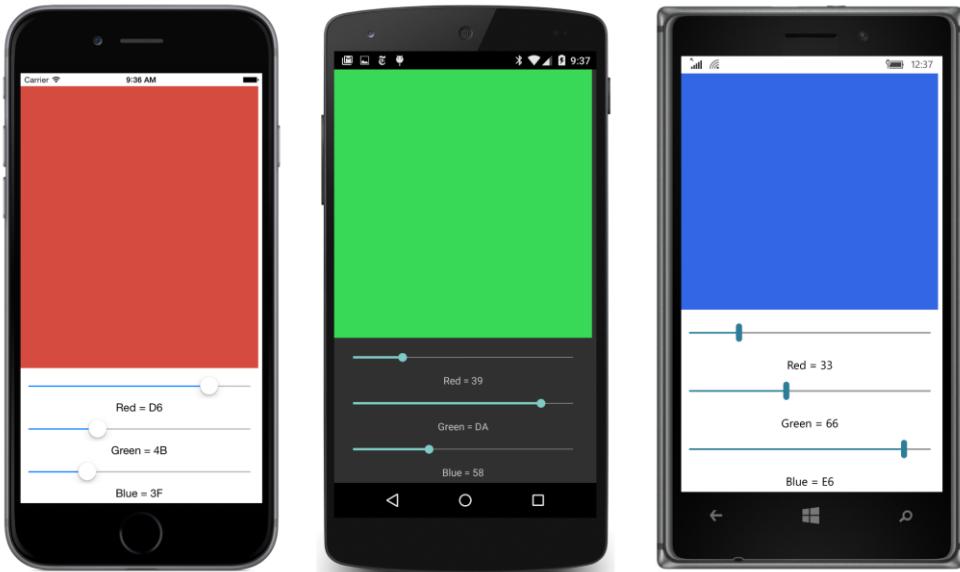
<StackLayout VerticalOptions = "CenterAndExpand" >
    <deslizador x: Nombre = "blueSlider" 
        ValueChanged = "OnSliderValueChanged" />

    <Etiqueta Texto = * {Binding Fuente = {x: blueSlider Referencia}},
        Path = Valor,
        Convertidor = {} StaticResource doubleToInt,
        ConverterParameter = 255,
        StringFormat = 'Azul = (0: X2)' * >

</StackLayout>
</StackLayout>
</Cuadrícula>
</Página de contenido>

```

Y aquí está el retrato de vista:



La disposición en el archivo XAML se prepara para el modo de paisaje en un par de maneras. Primero el Cuadrícula ya tiene una segunda columna. Esto significa que para cambiar al modo de paisaje, el archivo de código subyacente tiene que cambiar la altura de la segunda fila a cero y el ancho de la segunda columna a un valor distinto de cero.

En segundo lugar, la StackLayout que contiene toda la deslizador y Etiqueta elementos es accesible desde el código, ya que tiene un nombre, en concreto controlPanelStack. El archivo de código subyacente a continuación, puede hacer Grid.SetRow y Grid.SetColumn pide a este StackLayout para moverlo desde la fila 1 y la columna 0 a la fila 0 y la columna 1.

En el modo vertical, la BoxView tiene una altura de "*" (asterisco) y la StackLayout tiene una altura de Auto. ¿Quiere esto decir que el ancho de la StackLayout debiera ser Auto en modo horizontal? Eso no lo haría

ser prudente, ya que se reduciría el ancho de los deslizadores de elementos. Una mejor solución para el modo horizontal es dar tanto la BoxView y el StackLayout una anchura de “*” (asterisco) para dividir la pantalla en dos.

Aquí está el archivo de código subyacente que muestra el SizeChanged manejador en la página responsable de cambiar entre modo vertical y horizontal, así como la ValueChanged controlador para los deslizadores de elementos que establece el BoxView color:

```
pública clase parcial GridRgbSlidersPage : Pagina de contenido
{
    pública GridRgbSlidersPage ()
    {
        // Asegúrese de enlace a Toolkit biblioteca.
        nuevo Xamarin.FormsBook.Toolkit.DoubleToIntConverter ();

        InitializeComponent ();
    }

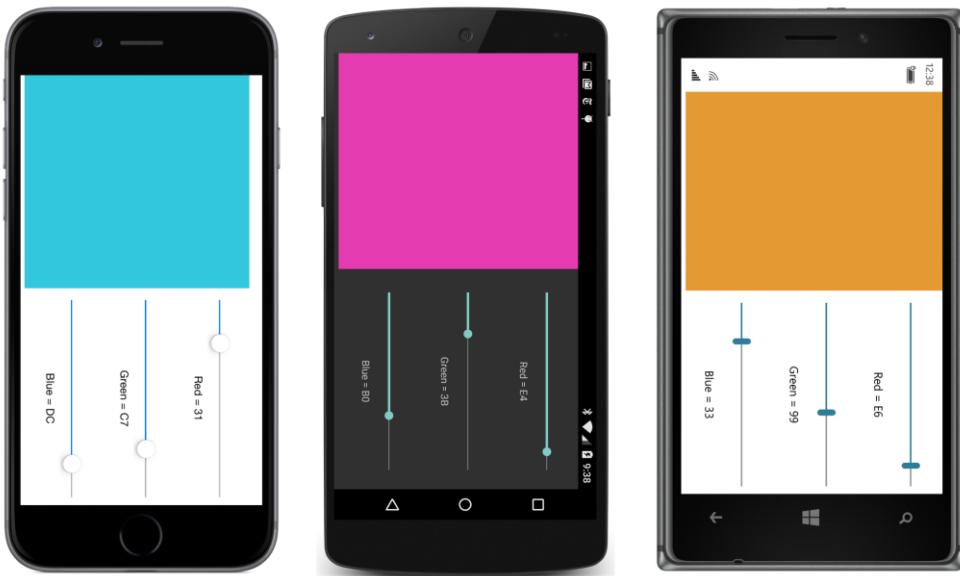
    vacío OnPageSizeChanged ( objeto remitente, EventArgs args )
    {
        // Modo retrato.
        Si (Anchura <Altura)
        {
            mainGrid.RowDefinitions [1].height = GridLength .Auto;
            mainGrid.ColumnDefinitions [1].Width = nuevo GridLength (0, GridUnitType .Absoluto);

            Cuadrícula .SetRow (controlPanelStack, 1);
            Cuadrícula .SetColumn (controlPanelStack, 0);
        }
        // Modo paisaje.
        más
        {
            mainGrid.RowDefinitions [1].height = nuevo GridLength (0, GridUnitType .Absoluto);
            mainGrid.ColumnDefinitions [1].Width = nuevo GridLength (1, GridUnitType .Estrella);

            Cuadrícula .SetRow (controlPanelStack, 0);
            Cuadrícula .SetColumn (controlPanelStack, 1);
        }
    }

    vacío OnSliderValueChanged ( objeto remitente, ValueChangedEventArgs args )
    {
        boxView.Color = nuevo Color (RedSlider.Value, greenSlider.Value, blueSlider.Value);
    }
}
```

Y aquí está el diseño de paisaje, que se muestra hacia los lados, como de costumbre:



Aviso, sobre todo en el iOS y Android pantallas, como cada par de deslizador y Etiqueta elementos se agrupan. Esto es consecuencia de una tercera vía que el archivo XAML se prepara para dar cabida a modo de paisaje. Cada par de deslizador y Etiqueta elementos se agrupan en un anidado StackLayout. Esto se da una VerticalOptions ajuste de CenterAndExpand para llevar a cabo esta separación.

Una se prestó poca atención a la organización de la BoxView y el panel de control: En el modo vertical, los dedos la manipulación de la deslizador elementos no oscurecer el resultado en el BoxView, y en modo horizontal, los dedos de los usuarios diestros no oscurecer la BoxView ya sea. (Por supuesto, los usuarios zurdos probablemente va a insistir en una opción de programa para cambiar los lugares!)

Las capturas de pantalla muestran la deslizador Los valores que aparecen en hexadecimal. Esto se hace con un enlace de datos, y que normalmente sería un problema. los Valor propiedad de la deslizador es de tipo doble, y si intenta formatear un doble con "X2" para hexadecimal, se produce una excepción. Un convertidor de tipos (llamado DoubleToIntConverter, por ejemplo) debe convertir la fuente doble a una Ent para el formato de cadenas. sin embargo, el deslizador elementos están configurados para una gama de 0 a 1, mientras que los valores enteros formateadas como hexadecimal deben variar de 0 a 255.

Una solución es hacer uso de la ConverterParameter propiedad de Unión. Lo que se establece en esta propiedad se pasa como el tercer argumento de la Convertir y ConvertBack métodos en el convertidor de valores. Aquí está la DoubleToIntConverter clase en el Xamarin.Forms biblioteca:

```
espacio de nombres Xamarin.FormsBook.Toolkit
{
    clase pública DoubleToIntConverter : IValueConverter
    {
        objeto público Convertir( objeto valor, Tipo tipo de objetivo,
            objeto parámetro, CultureInfo cultura)
        {

```

```

cuerda StrParam = parámetro como cuerda ;
doble multiplicador = 1;

Si (I Cuerda .IsNullOrEmpty (StrParam))
{
    Doble .TryParse (StrParam, fuera multiplicador);
}

regreso (En t) Mates .Redondo((doble) * Valor de multiplicador);
}

objeto público ConvertBack (objeto valor, Tipo tipo de objetivo,
                            objeto parámetro, CultureInfo cultura)
{
    cuerda StrParam = parámetro como cuerda ;
    doble divisor = 1;

    Si (I Cuerda .IsNullOrEmpty (StrParam))
    {
        Doble .TryParse (StrParam, fuera divisor);
    }

    regreso (En t) Valor / divisor;
}
}
}
}

```

los Convertir y ConvertBack métodos suponen que la parámetro argumento es una cadena y, si es así, tratar de convertirlo en una doble. Este valor se multiplica por el doble valor siendo convertido, y después el producto se convierte en una En t.

La combinación del convertidor de valor, el parámetro de convertidor, y la cadena de formato convierte los valores que van de 0 a 1 que viene de la deslizador a números enteros en el intervalo de 0 a 255 que luego son formateado como dos dígitos hexadecimales:

```

<Etiqueta Texto = "(Binding Fuente = {x: redSlider Referencia}),
Path = Valor,
Convertidor = {} StaticResource doubleToInt,
ConverterParameter = 255,
StringFormat = 'Red = {0: X2}'>
```

Por supuesto, si estuviera definiendo la Unión en el código, es probable que establezca el ConverterParameter propiedad en el valor numérico de 255 en lugar de una serie de "255", y la lógica en el duplicó con doubleToIntConverter fallaría. convertidores de valores simples son generalmente más simple de lo que debería ser para prueba de balas completa.

¿Puede un programa como GridRgbSliders ser realizado completamente sin la deslizador controladores de eventos en el archivo de código subyacente? Código sin duda seguirá siendo necesaria, pero algunos de ellos se aleja de la lógica UserInterface. Ese es el objetivo principal de la arquitectura Model-View-ViewModel explorado en el capítulo siguiente.

capítulo 18

MVVM

¿Puede usted recordar sus primeras experiencias con la programación? Es probable que su principal objetivo era sólo conseguir el funcionamiento del programa y, a continuación, conseguir que funcione correctamente. Probablemente no piensa mucho acerca de la organización o estructura del programa. Eso fue algo que vino después.

La industria de la informática en su conjunto ha pasado por una evolución similar. Como desarrolladores, que ahora todos damos cuenta de que una vez que una aplicación comienza a crecer en tamaño, por lo general es una buena idea de imponer algún tipo de estructura o arquitectura en el código. La experiencia con este proceso sugiere que a menudo es mejor empezar a pensar en esta arquitectura tal vez antes de cualquier código está escrito en absoluto. En la mayoría de los casos, una estructura de programa deseable esfuerza por lograr una "separación de las preocupaciones" por la que diferentes piezas del programa se centran en diferentes tipos de tareas.

En un programa gráficamente interactiva, una técnica obvia es separar la interfaz de usuario de la lógica no interfaz de usuario subyacente, a veces llamado *lógica de negocios*. La primera descripción formal de este tipo de arquitectura para interfaces gráficas de usuario se llama Modelo-Vista-Controlador (MVC), pero ya que esta arquitectura ha dado lugar a otros derivados de ella.

En cierta medida, la naturaleza de la propia interfaz de programación influye en la arquitectura de la aplicación. Por ejemplo, una interfaz de programación que incluye un lenguaje de marcado con enlaces de datos podría sugerir un modo particular de estructurar una aplicación.

En efecto, existe un modelo arquitectónico que fue diseñado específicamente con XAML en mente. Esto se conoce como Modelo-Vista-ViewModel o MVVM. Este capítulo trata sobre los fundamentos de MVVM (incluyendo la interfaz de comandos), pero se podrán ver más sobre MVVM en el capítulo siguiente, que abarca puntos de vista de recolección. Además, algunas otras características de Xamarin.Forms se utilizan a menudo en conjunción con MVVM; estas características incluyen *disparadores y comportamientos*, y son el tema del capítulo 23.

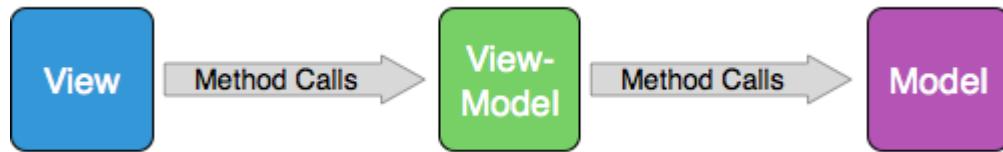
interrelaciones MVVM

MVVM divide una aplicación en tres capas:

- El modelo proporciona datos subyacentes, a veces con archivo o página web accesos.
- El modelo de vista conecta el modelo y la vista. Ayuda a gestionar los datos del modelo para que sea más susceptible a la vista, y viceversa.
- La vista es la capa de interfaz de usuario o la presentación, implementado generalmente en XAML.

El modelo ignora el modelo de vista. En otras palabras, el modelo no sabe nada sobre el público

propiedades y métodos del modelo de vista, y sin duda nada sobre su funcionamiento interno. Del mismo modo, el modelo de vista es ignorante de la vista. Si toda la comunicación entre las tres capas se produce a través de llamadas a métodos y accede a la propiedad, a continuación, llama en una sola dirección se permiten. La vista sólo hace llamadas en el modelo de vista o accede a las propiedades del modelo de vista, y el modelo de vista de manera similar sólo hace llamadas en el modelo o accesos, Características del modelo:



Estas llamadas a métodos permiten la vista para obtener información del modelo de vista, que a su vez obtiene información del modelo.

En ambientes modernos, sin embargo, los datos son a menudo dinámico. A menudo, el modelo obtendrá más o nuevos datos que deben ser comunicada al modelo de vista y, finalmente, a la vista. Por esta razón, la vista puede asociar controladores de eventos que se implementan en el modelo de vista, y el modelo de vista puede asociar controladores de eventos definidos por el modelo. Esto permite la comunicación bidireccional sin dejar de ocultar la vista del modelo de vista, y el modelo de vista del modelo:



MVVM fue diseñado para tomar ventaja de XAML y enlaces de datos en particular basados en XAML. En general, la vista es una clase de página que utiliza XAML para construir la interfaz de usuario. Por lo tanto, la conexión entre la vista y el modelo de vista consiste en gran parte, y tal vez exclusivamente de enlaces de datos basadas en XAML:



Los programadores que son muy apasionados por MVVM a menudo tienen un objetivo informal de expresar todas las interacciones entre la vista y el modelo de vista en una clase de página con enlaces de datos basadas en XAML, y en el proceso de reducir el código en la página del archivo a un simple código subyacente `InitializeComponent`

llamada. Este objetivo es difícil de lograr en la programación de la vida real, pero es un placer cuando sucede.

Pequeños programas -tales como los de un libro como éste, a menudo se hacen más grandes cuando se introduce MVVM. No deje que esto desalienta su uso de MVVM! Utilice los ejemplos aquí para ayudarle a determinar cómo

MVVM se puede utilizar en un programa más amplio, y que finalmente va a ver que ayuda enormemente en la arquitectura de sus aplicaciones.

ViewModels y los datos de enlace

En muchas manifestaciones bastante simples de MVVM, el modelo está ausente o es solamente implícita, y el modelo de vista contiene toda la lógica de negocio. La vista y el modelo de vista comunican a través de enlaces de datos XAMLbased. Los elementos visuales en la vista son objetivos-enlace de datos, y propiedades en el modelo de vista son los datos de unión a fuentes.

Idealmente, un modelo de vista debe ser independiente de cualquier plataforma en particular. Esta independencia permite ViewModels a ser compartidos entre otros entornos basados en XAML (como Windows), además de Xamarin.Forms. Por esta razón, usted debe tratar de evitar el uso de la siguiente declaración en sus ViewModels:

[utilizando Xamarin.Forms;](#)

Esa regla se rompe con frecuencia en este capítulo! Uno de los ViewModels se basa en la Xamarin.Forms

[Color estructura, y otros usos Device.StartTimer. Así que vamos a llamar a la evitación de un tema específico sobre Xamarin.Forms en el](#) modelo de vista una “sugerencia” en lugar de una “regla”.

Los elementos visuales en la vista califican como objetivos de enlace de datos debido a que las propiedades de estos elementos visuales están respaldados por propiedades enlazables. Para ser una fuente de enlace de datos, un modelo de vista debe implementar un protocolo de notificación para indicar cuándo una propiedad en el modelo de vista ha cambiado. Este protocolo es la notificación [INotifyPropertyChanged interfaz](#), que se define en el System.ComponentModel-

Modelo espacio de nombres muy simple con un solo evento:

```
interfaz pública INotifyPropertyChanged
{
    evento PropertyChangedEventHandler PropertyChanged;
}
```

los [INotifyPropertyChanged interfaz](#) es tan central para MVVM que en las discusiones informales de la interfaz es a menudo abreviado INPC.

Los PropertyChanged evento en el INotifyPropertyChanged interfaz es de tipo Propiedad-Cambiado-manejador de sucesos. Un controlador de este PropertyChanged controlador de eventos recibe una instancia de la PropertyChangedEventArgs clase, que define una sola propiedad llamada Nombre de la propiedad de tipo cuerda indicando lo que la propiedad en el modelo de vista ha cambiado. El controlador de eventos puede entonces tener acceso a esa propiedad.

Una clase que implementa INotifyPropertyChanged debe disparar una PropertyChanged evento cada vez que una propiedad cambia públicos, pero la clase debe *no* desencadenar el evento cuando la propiedad se limitó a exponer, pero no cambió.

Algunas clases definen propiedades-propiedades que se inicializan en el constructor y luego nunca cambian inmutables. Estas propiedades no necesitan disparar PropertyChanged eventos debido a que una Apuntalar-ertyChanged controlador se puede conectar sólo después de que el código en el constructor termina, y las propiedades inmutables nunca cambian después de ese tiempo.

En teoría, una clase ViewModel se puede derivar de bindableObject y poner en práctica sus propiedades públicas como BindableProperty objetos. bindableObject implementos INotifyPropertyChanged y se dispara automáticamente una PropertyChanged evento cuando cualquier propiedad respaldado por una BindableProperty cambios. Pero se deriva de bindableObject es una exageración para un modelo de vista. Porque bindableObject y BindableProperty son específicos de Xamarin.Forms, un modelo de vista tal es la plataforma ya no es independiente, y la técnica no proporciona ningún ventajas reales sobre una implementación simple de INotifyPropertyChanged.

Un reloj modelo de vista

Supongamos que usted está escribiendo un programa que necesita acceso a la fecha y hora actuales, y desea utilizar esa información a través de enlaces de datos. La biblioteca de clases base de .NET proporciona la fecha y hora a través de la información Fecha y hora estructura. Para obtener la fecha y hora actuales, sólo el acceso al DateTime.Now propiedad. Esa es la forma habitual de escribir una aplicación de reloj.

Sin embargo, para los propósitos de enlace de datos, Fecha y hora tiene un defecto grave: Proporciona información estática simplemente sin notificación cuando la fecha o el tiempo ha cambiado.

En el contexto de MVVM, la Fecha y hora estructura quizás califica como un modelo en el sentido de que Fecha y hora proporciona todos los datos que necesitamos, pero no en una forma que es propio para enlaces de datos. Es necesario escribir un modelo de vista que hace uso de Fecha y hora sino que proporciona notificaciones cuando la fecha o el tiempo ha cambiado.

los **Xamarin.FormsBook.Toolkit** la biblioteca contiene DateTimeViewModel clase se muestra a continuación. La clase tiene sólo una propiedad, que se nombra Fecha y hora de tipo Fecha y hora, pero esta propiedad cambia de forma dinámica como resultado de llamadas frecuentes a DateTime.Now en un Device.StartTimer llamar de vuelta.

Observe que el DateTimeViewModel clase se basa en el INotifyPropertyChanged interfaz e incluye una utilizando Directiva para la System.ComponentModel espacio de nombres que define esta interfaz. Para implementar esta interfaz, la clase define un evento público denominado PropertyChanged.

¡Cuidado: Es muy fácil de definir una PropertyChanged evento en su clase sin especificar explícitamente que los implementa la clase INotifyPropertyChanged! Las notificaciones serán ignorados si no se especifica explícitamente que la clase se basa en la INotifyPropertyChanged interfaz:

```
utilizando Sistema;
utilizando System.ComponentModel;
utilizando Xamarin.Forms;

espacio de nombres Xamarin.FormsBook.Toolkit
{
    clase pública DateTimeViewModel : INotifyPropertyChanged
```

```

{
    Fecha y hora fechaHora = Fecha y hora .Ahora;

    evento público PropertyChangedEventHandler PropertyChanged;

    público DateTimeViewModel ()
    {
        Dispositivo .StartTimer ( Espacio de tiempo .FromMilliseconds (15), OnTimerTick);
    }

    bool OnTimerTick ()
    {
        DateTime = Fecha y hora .Ahora;
        return true ;
    }

    público Fecha y hora Fecha y hora
    {
        conjunto privado
        {
            Si (FechaHoral = valor )
            {
                fechaHora = valor ;

                // desencadenar el evento.
                PropertyChangedEventHandler handler = PropertyChanged;

                Si (Manejador! = nulo )
                {
                    entrenador de animales( esta , nuevo PropertyChangedEventArgs ( "Fecha y hora" ));
                }
            }
        }

        obtener
        {
            regreso fecha y hora;
        }
    }
}
}

```

La única propiedad pública en esta clase se llama Fecha y hora de tipo Fecha y hora, y se asocia con un campo respaldo privado llamado fecha y hora. Las propiedades públicas en ViewModels suelen tener campos de respaldo privadas. Los conjunto descriptor de acceso de la Fecha y hora es propiedad privada a la clase, y se actualiza cada 15 milisegundos de la devolución de llamada del temporizador.

Aparte de eso, el conjunto de acceso se construye de una manera muy estándar para ViewModels: Se comprueba en primer lugar si el valor que se establece en la propiedad es diferente de la fecha y hora campo de respaldo. Si no es así, se establece que el campo respaldo del valor de entrada y dispara el PropertyChanged manipulador con el nombre de la propiedad. Se considera muy mala práctica para disparar el PropertyChanged si el manejador

meramente propiedad está siendo ajustado a su valor actual, y que incluso podría conducir a problemas que afectan a los ciclos infinitos de valores de propiedades recursivas de dos vías fijaciones.

Este es el código de la conjunto de acceso que desencadena el evento:

```
PropertyChangedEventHandler handler = PropertyChanged;

Si (Manejador! = nulo )
{
    entrenador de animales( esta , nuevo PropertyChangedEventArgs ("Fecha y hora"));
}
```

Esa forma es preferible codificar como este, que no guarda el controlador en una variable independiente:

```
Si (PropertyChanged! = nulo )
{
    PropertyChanged ( esta , nuevo PropertyChangedEventArgs ("Fecha y hora"));
}
```

En un entorno multiproceso, una PropertyChanged manejador puede ser separado entre el Si declaración que comprueba una nulo valor y el disparo real del evento. Guardar el controlador en una variable independiente evita que causen un problema, por lo que es un buen hábito de adoptar incluso si aún no está trabajando en un entorno multiproceso.

los obtener de acceso, simplemente devuelve el fecha y hora campo de respaldo.

los **MvvmClock** programa demuestra cómo el **DateTimeViewModel** clase es capaz de proporcionar información de fecha y hora actualizada a la interfaz de usuario a través de enlaces de datos:

```
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    xmlns: sys = " clr-espacio de nombres: System; montaje = mscorelib "
    xmlns: kit de herramientas =
        " clr-espacio de nombres: Xamarin.FormsBook.Toolkit; montaje = Xamarin.FormsBook.Toolkit "
    x: Class = " MvvmClock.MvvmClockPage " >

    < ContentPage.Resources >
        < ResourceDictionary >
            < kit de herramientas: DateTimeViewModel x: Key = " dateTimeViewModel " />

            < Estilo Tipo de objetivo = " Etiqueta " >
                < Setter Propiedad = " Tamaño de fuente " Valor = " Grande " />
                < Setter Propiedad = " HorizontalTextAlignment " Valor = " Centrar " />
            </ Estilo >
        </ ResourceDictionary >
    </ ContentPage.Resources >

    < StackLayout VerticalOptions = " Centrar " >
        < Etiqueta Texto = " {Binding fuente = {x: Estático: sys}, DateTime.Now
            StringFormat = 'Este programa se inició en {0: F} } " />
        < Etiqueta Texto = " Pero ahora... " />
    </ StackLayout >

```

```
< Etiqueta Texto = " {Binding Fuente = () StaticResource dateViewModel,
    Path = DateTime.Hour,
    StringFormat = 'La hora es {0}' } >

< Etiqueta Texto = " {Binding Fuente = () StaticResource dateViewModel,
    Path = DateTime.Minute,
    StringFormat = 'El minuto es {0}' } >

< Etiqueta Texto = " {Binding Fuente = () StaticResource dateViewModel,
    Path = DateTime.Second,
    StringFormat = 'Los segundos son {0}' } >

< Etiqueta Texto = " {Binding Fuente = () StaticResource dateViewModel,
    Path = DateTime.Millisecond,
    StringFormat = 'Los milisegundos son {0}' } >

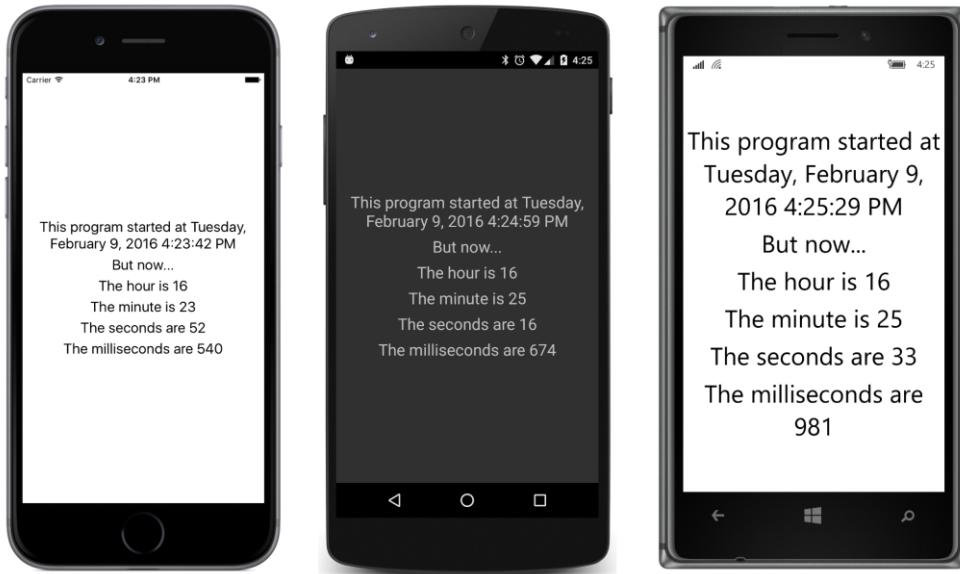
</ StackLayout >
</ Página de contenido >
```

los recursos la sección de la página de la instancia DateTimeViewModel y también define un implícito Estilo Para el Etiqueta.

El primero de los seis Etiqueta elementos establece su Texto propiedad a un Unión objeto que implica el .NET real Fecha y hora estructura. los Fuente la propiedad de que la unión es una x: Estático extensión de marcado que hace referencia a la estática DateTime.Now propiedad para obtener la fecha y la hora cuando el programa se inicia por primera ejecución. No Camino se requiere en esta unión. La especificación de formato "F" es para el patrón de fecha / tiempo completo, con versiones largas de las cadenas de fecha y hora. Aunque esto Etiqueta muestra la fecha y la hora cuando el programa se pone en marcha, nunca se actualizará.

Los últimos cuatro enlaces de datos *será* estar actualizado. En estos enlaces de datos, la Fuente propiedad se establece una StaticResource extensión de marcado que hace referencia a la DateTimeViewModel objeto. los Camino se establece en diversas subpropiedades de la Fecha y hora propiedad de ese modelo de vista. Detrás de las escenas, la infraestructura de unión se une un controlador en el PropertyChanged evento en el DateTimeViewModel. Este controlador comprueba para un cambio en el Fecha y hora la propiedad y los cambios Texto propiedad de la Etiqueta siempre que los cambios de propiedad.

El archivo de código subyacente está vacía a excepción de una InitializeComponent llamada. Los enlaces de datos de los últimos cuatro etiquetas muestran una hora actualizada que cambia tan rápido como la frecuencia de actualización de video:



El margen de beneficio en este archivo XAML puede simplificarse mediante el establecimiento de la `BindingContext` propiedad de la `StackLayout` a una `StaticResource` extensión de marcado que hace referencia al modelo de vista. Ese `EnlazamientoContext` se propaga a través del árbol visual para que pueda retirar el `Fuente` configuración de la final a cuatro `Etiqueta` elementos:

```
< StackLayout VerticalOptions = "Centrar"
    BindingContext = "{ StaticResource dateViewModel } >

    < Etiqueta Texto = "{ Binding fuente = {x: Estático: sys}, DateTime.Now
        StringFormat = "Este programa se inició en {0: F} } " />

    < Etiqueta Texto = "Pero ahora... " />

    < Etiqueta Texto = "{ Binding Path = DateTime.Hour,
        StringFormat = "La hora es {0} } " />

    < Etiqueta Texto = "{ Binding Path = DateTime.Minute,
        StringFormat = "El minuto es {0} } " />

    < Etiqueta Texto = "{ Binding Path = DateTime.Second,
        StringFormat = "Los segundos son {0} } " />

    < Etiqueta Texto = "{ Binding Path = DateTime.Millisecond,
        StringFormat = "Los milisegundos son {0} } " />
</ StackLayout >
```

los Unión en la primera `Etiqueta` anula que `BindingContext` con su propia `Fuente` ajuste.

Incluso se puede eliminar el `DateTimeViewModel` elemento de la `ResourceDictionary` y instanciarlo justo en el `StackLayout` Entre `BindingContext` etiquetas de propiedad de elementos:

```

< StackLayout VerticalOptions = "Centrar" >
    < StackLayout.BindingContext >
        < !kit de herramientas: DateTimeViewModel />
    </ StackLayout.BindingContext >

    < Etiqueta Texto = " {Binding fuente = {x: Estático: sys}, DateTime.Now
        StringFormat = 'Este programa se inició en {0: F}' } " />

    < Etiqueta Texto = " Pero ahora... " />

    < Etiqueta Texto = " {Binding Path = DateTime.Hour,
        StringFormat = 'La hora es {0}' } " />

    < Etiqueta Texto = " {Binding Path = DateTime.Minute,
        StringFormat = 'El minuto es {0}' } " />

    < Etiqueta Texto = " {Binding Path = DateTime.Second,
        StringFormat = 'Los segundos son {0}' } " />

    < Etiqueta Texto = " {Binding Path = DateTime.Millisecond,
        StringFormat = 'Los milisegundos son {0}' } " />
</ StackLayout >

```

O bien, se puede establecer el BindingContext propiedad de la StackLayout a una Unión que incluye la Fecha y hora propiedad. los BindingContext a continuación, se convierte en el Fecha y hora valor, que permite a los enlaces individuales para simplemente Referencia propiedades del .NET Fecha y hora estructura:

```

< StackLayout VerticalOptions = "Centrar" 
    BindingContext = " {Binding Fuente = {} StaticResource dateViewModel,
        Path = DateTime} " >

    < Etiqueta Texto = " {Binding fuente = {x: Estático: sys}, DateTime.Now
        StringFormat = 'Este programa se inició en {0: F}' } " />

    < Etiqueta Texto = " Pero ahora... " />

    < Etiqueta Texto = " {Binding Path = horas,
        StringFormat = 'La hora es {0}' } " />

    < Etiqueta Texto = " {Binding Path = minuto,
        StringFormat = 'El minuto es {0}' } " />

    < Etiqueta Texto = " {Path = Encuadernación En segundo lugar,
        StringFormat = 'Los segundos son {0}' } " />

    < Etiqueta Texto = " {Binding Path = milisegundos,
        StringFormat = 'Los milisegundos son {0}' } " />
</ StackLayout >

```

Es posible que tenga dudas de que esto va a funcionar! Detrás de las escenas, una unión que normalmente los datos instala una Apuntalar-
ertyChanged controlador de eventos y los relojes de las propiedades particulares que se están cambiados, pero no puede en este caso porque la fuente de los enlace de datos es una Fecha y hora valor, y Fecha y hora no implementa
INotifyPropertyChanged. sin embargo, el BindingContext de estos Etiqueta Elementos de cambios con

cada cambio en el Fecha y hora propiedad en el modelo de vista, por lo que la infraestructura de la unión accede a nuevos valores de estas propiedades en ese momento.

Como los enlaces individuales en el Texto propiedades disminuyen en longitud y complejidad, se puede quitar la Camino nombre de atributo y poner todo en una sola línea y nadie se confunda:

```
< StackLayout VerticalOptions = "Centrar" >
    < StackLayout.BindingContext >
        < Unión Camino = "Fecha y hora" >
            < Binding.Source >
                < kit de herramientas: DateTimeViewModel />
            </ Binding.Source >
        </ Unión >
    </ StackLayout.BindingContext >

    < Etiqueta Texto = "(Binding fuente = {x: Estático: sys}, DateTime.Now
        StringFormat = 'Este programa se inició en {0: F}') " />

    < Etiqueta Texto = "Pero ahora..." />

    < Etiqueta Texto = "(Binding horas, StringFormat = 'La hora es {0}') " />
    < Etiqueta Texto = "(Minute Binding, StringFormat = 'El minuto es {0}') " />
    < Etiqueta Texto = "(Binding En segundo lugar, StringFormat = 'Los segundos son {0}') " />
    < Etiqueta Texto = "(Binding Milisegundo, StringFormat = 'Los milisegundos son {0}') " />
</ StackLayout >
```

En futuros programas de este libro, los enlaces individuales en su mayoría ser tan corto y tan elegante como sea posible.

propiedades interactivas en un ViewModel

El segundo ejemplo de un modelo de vista hace algo tan básico que nunca escribiría un modelo de vista para este propósito. Los SimpleMultiplierViewModel la clase se limita a multiplicar dos números juntos. Pero es un buen ejemplo para demostrar la sobrecarga y la mecánica de un modelo de vista que tiene varias propiedades interactivas. (Y a pesar de que nunca iba a escribir un modelo de vista para multiplicar dos números, se podría escribir un modelo de vista para resolver ecuaciones cuadráticas o algo mucho más complejo.)

Los SimpleMultiplierViewModel clase es parte de la SimpleMultiplier proyecto:

```
utilizando Sistema;
utilizando System.ComponentModel;

espacio de nombres SimpleMultiplier
{
    clase SimpleMultiplierViewModel : INotifyPropertyChanged
    {
        doble multiplicando, multiplicador, producto;

        evento público PropertyChangedEventHandler PropertyChanged;
```

```
pública doble Multiplicando
{
    conjunto
    {
        Si (Multiplicando! = valor )
        {
            multiplicando = valor ;
            OnPropertyChanged ( "Multiplicando" );
            UpdateProduct ();
        }
    }
    obtener
    {
        regreso multiplicando;
    }
}

pública doble Multiplicador
{
    conjunto
    {
        Si (Multiplicador! = valor )
        {
            multiplicador = valor ;
            OnPropertyChanged ( "Multiplicador" );
            UpdateProduct ();
        }
    }
    obtener
    {
        regreso multiplicador;
    }
}

pública doble Producto
{
    conjunto protegido
    {
        Si (Producto = valor )
        {
            producto = valor ;
            OnPropertyChanged ( "Producto" );
        }
    }
    obtener
    {
        regreso producto;
    }
}

vacío UpdateProduct ()
{
    Producto = Multiplicando * multiplicador;
}
```

```
protected void OnPropertyChanged ( cuerda nombre de la propiedad)
{
    PropertyChangedEventHandler handler = PropertyChanged;

    Si (Manejador != nulo )
    {
        PropertyChanged ( esta ,nuevo PropertyChangedEventArgs (nombre de la propiedad));
    }
}
```

La clase define tres propiedades públicas de tipo doble, llamado Multiplicando, multiplicador, y Producto. Cada propiedad está respaldada por un campo privado. los conjunto y obtener descriptores de acceso de las dos primeras propiedades son públicas, pero el conjunto descriptor de acceso de la Producto se protege la propiedad para evitar que sea establecido fuera de la clase al tiempo que permite una clase descendiente para cambiarlo.

los conjunto descriptor de acceso de cada propiedad comienza comprobando si el valor de la propiedad es en realidad cambiando, y si es así, se establece el campo respaldo a ese valor y llama a un método llamado `OnPropertyChanged` con ese nombre de propiedad.

los `INotifyPropertyChanged` interfaz no requiere una `OnPropertyChanged` clases de método, pero a menudo incluyen un `ViewModel` para reducir la repetición de código. Por lo general se define como `Property` en caso de que necesite para derivar un modelo de vista de otro y desencadenar el evento en la clase derivada. Más adelante en este capítulo, verá técnicas de reducir la repetición de código en `INotifyPropertyChanged` clases aún más.

los conjunto descriptores de acceso tanto para el Multiplicando y Multiplicador propiedades concluyen llamando al UpdateProduct método. Este es el método que realiza la tarea de multiplicar los valores de las dos propiedades y establecer un nuevo valor para el Producto propiedad, que a su vez dispara su propio Propiedad-cambiado evento.

Aquí está el archivo XAML que hace uso de este modelo de vista:

```
< Pagina de contenido xmlns = "http://xamarin.com/schemas/2014/forms" >
    xmlns: x = "http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns: locales = "CLR-espacio de nombres: SimpleMultiplier"
    x: Class = "SimpleMultiplier.SimpleMultiplierPage"
    Relleno = "10,0" >

< ContentPage.Resources >
    < ResourceDictionary >
        < locales: SimpleMultiplierViewModel x: Key = "viewmodel" />

        < Estilo Tipo de objetivo = "Etiqueta" >
            <Setter Propiedad = "Tamaño de fuente" Valor = "Grande" />
        </ Estilo >

    </ ResourceDictionary >
</ ContentPage.Resources >
```

```

< StackLayout BindingContext = " { StaticResource viewmodel } >

    < StackLayout VerticalOptions = " CenterAndExpand " >
        < deslizador Valor = " Multiplicando {Binding} " />
        < deslizador Valor = " {StaticResource La unión Multiplicador} " />
    </ StackLayout >

    < StackLayout Orientación = " Horizontal " >
        Espaciado = " 0 "
        VerticalOptions = " CenterAndExpand "
        HorizontalOptions = " Centrar " >
            < Etiqueta Texto = " {Binding multiplicando, StringFormat = '(0: F3)'} " />
            < Etiqueta Texto = " {Binding multiplicador, StringFormat = 'x (0: F3)'} " />
            < Etiqueta Texto = " {Binding producto, StringFormat = '= (0: F3)'} " />
        </ StackLayout >
    </ StackLayout >
</ Página de contenido >

```

los SimpleMultiplierViewModel se crea una instancia en el recursos diccionario y ajustado a la BindingContext propiedad de la StackLayout mediante el uso de una StaticResource extensión de marcado. Ese BindingContext es heredado por todos los hijos y nietos de la StackLayout, que incluye dos deslizador y tres Etiqueta elementos. El uso de la BindingContext permite que estos enlaces sean lo más simple posible.

El modo por defecto de la unión Valor propiedad de la deslizador es TwoWay. Los cambios en el Valor propiedad de cada deslizador causar cambios en las propiedades del modelo de vista.

El tres Etiqueta elementos de visualización de los valores de las tres propiedades del ViewModel con algún formato que inserta veces y es igual a las muestras con los números:

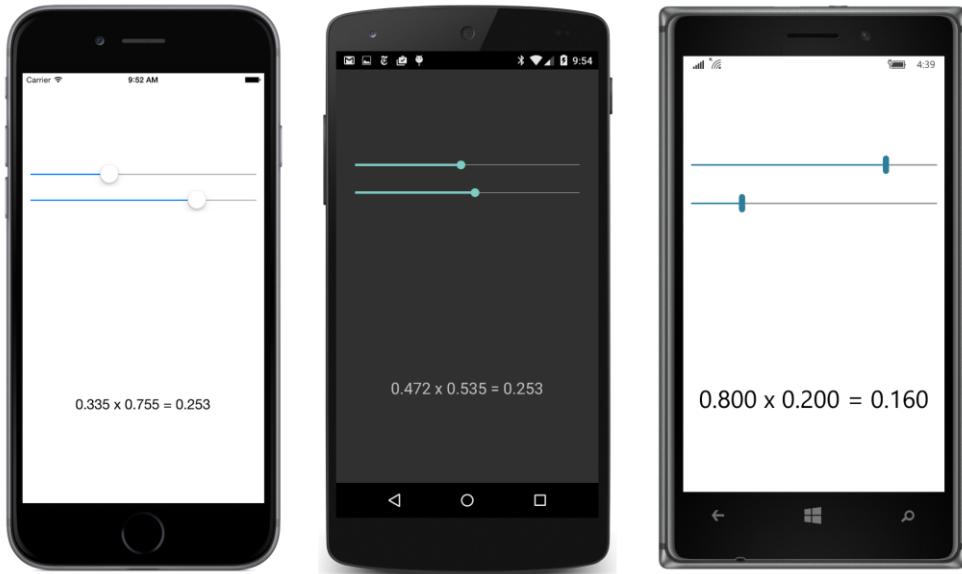
```

< Etiqueta Texto = " {Binding multiplicando, StringFormat = '(0: F3)'} " />
< Etiqueta Texto = " {Binding multiplicador, StringFormat = 'x (0: F3)'} " />
< Etiqueta Texto = " {Binding producto, StringFormat = '= (0: F3)'} " />

```

Para los dos primeros, se puede enlazar, alternativamente, la Texto propiedad de la Etiqueta elementos directamente a la Valor propiedad de los correspondientes deslizador. Pero eso requeriría que usted da a cada uno deslizador con un nombre x: Nombre y hacer referencia a ese nombre en una Fuente argumento mediante el uso de la x: Referencia extensión de marcado. El enfoque utilizado en este programa es mucho más limpio y verifica que los datos es hacer un viaje completo a través del modelo de vista de cada uno deslizador a cada Etiqueta.

No hay nada en el archivo de código subyacente, excepto una llamada a InitializeComponent en el constructor. Toda la lógica de negocios se encuentra en el modelo de vista, y toda la interfaz de usuario se define en XAML:



Si desea, puede inicializar el modelo de vista, ya que se crea una instancia en el recursos diccionario:

```
< locales: SimpleMultiplierViewModel x: Key = "viewmodel "
    Multiplicando = "0.5"
    Multiplicador = "0.5" />
```

los deslizador elementos conseguirán estos valores iniciales como resultado de la unión de dos vías.

La ventaja de separar la interfaz de usuario de la lógica de negocio subyacente se hace evidente cuando se desea cambiar la interfaz de usuario algo, tal vez mediante la sustitución de una paso a paso Para el deslizador para uno o ambos números:

```
< StackLayout VerticalOptions = "CenterAndExpand" >
    < deslizador Valor = "Multiplicando {Binding}" />
    < paso a paso Valor = "{ La unión Multiplicador }" />
</ StackLayout >
```

Aparte de las diferentes gamas de los dos elementos, la funcionalidad es idéntica:



También podría sustituir una Entrada:

```
< StackLayout VerticalOptions = "CenterAndExpand " >
    < deslizador Valor = " Multiplicando (Binding) " />
    < Entrada Texto = " @ La unión Multiplicador " />
</ StackLayout >
```

El modo de unión por el impago Texto propiedad de la Entrada es también TwoWay, todo lo que tiene que preocuparse es la conversión entre la propiedad de origen doble y la propiedad de destino cuerda. Afortunadamente, esta conversión se realiza automáticamente por la infraestructura de la unión:



Si escribe una serie de caracteres que no se pueden convertir en una doble, la unión mantendrá el último valor válido. Si quieres una validación más sofisticado, que tendrá que implementar su propio (por ejemplo con un disparador, que será discutido en el capítulo 23).

Un experimento interesante es que escribir **1E-1**, la cual es la notación científica que se puede convertir en una dupló con **ble**. Verás que cambia inmediatamente a **"0,1"** en el Entrada. Este es el efecto de la **TwoWay** vinculante: El **Multiplicador** propiedad se establece en **1E-1** de la Entrada pero el **Encadenar** método que la infraestructura de la unión llamadas cuando el valor vuelve a la Entrada devuelve el texto **"0,1"**. Debido a que es diferente de la existente Entrada texto, se establece el nuevo texto. Para evitar que esto ocurra, se puede establecer el modo de unión a **OneWayToSource**:

```
<StackLayout VerticalOptions = "CenterAndExpand">
    <deslizador Valor = "Multiplicando (Binding)" />
    <Entrada Texto = "(Binding multiplicador, Mode = OneWayToSource)" />
</StackLayout>
```

Ahora el **Multiplicador** propiedad del modelo de vista se ajusta desde el **Texto** propiedad de la Entrada, pero no a la inversa. Si usted no necesita estos dos puntos de vista que actualizarse desde el modelo de vista, se puede establecer a ambos a **OneWayToSource**. Pero en general, usted querrá fijaciones MVVM sean **TwoWay**.

Debe preocuparse acerca de los ciclos infinitos en dos vías fijaciones? Por lo general no, porque Propiedad-cambiado eventos se activan sólo cuando la propiedad cambia de realidad y no cuando es meramente establece en el mismo valor. En general, el origen y el destino dejarán de actualizar entre sí después de un rebote o dos. Sin embargo, es posible escribir un convertidor de valores "patológico" que no provee para las conversiones de ida y vuelta, y que de hecho podría causar infinitos ciclos de actualización de dos vías fijaciones.

Un modelo de vista del color

El color siempre ofrece un buen medio para explorar las características de una interfaz gráfica de usuario, por lo que probablemente no se sorprenderá al saber que la **Xamarin.FormsBook.Toolkit** librería contiene una clase llamada **ColorViewModel**.

Los **ColorViewModel** clase expone una **Color** propiedad, sino también **Rojo**, **verde**, **azul**, **alfa**, **Hue**, **Saturación**, y **Luminosidad** propiedades, todos los cuales se pueden ajustar de forma individual. Esta no es una característica que la **Xamarin.Form Color** estructura proporciona. Una vez **Color** el valor es creado a partir de una **Color** constructor o uno de los métodos en **Color** comenzando con las palabras **Añadir**, **De**, **Multiplicar**, o **Con**, es inmutable.

Esta **ColorViewModel** clase se complica por la interrelación de su **Color** propiedad y todas las propiedades de los componentes. Por ejemplo, supongamos que el **Color** propiedad se establece. La clase debe disparar una **Apuntalar-Changed** guía no sólo para **Color** sino también para cualquier componente (tal como rojo o Matiz) que también cambia. Del mismo modo, si el rojo cambios de propiedad, entonces la clase deben disparar una **PropertyChanged** acontecimiento para los dos **rojo** y **Color**, y probablemente La saturación de color, y Luminosidad también.

Los **ColorViewModel** clase resuelve este problema mediante el almacenamiento de un campo de respaldo para la **Color** propiedad única. Todos conjunto descriptores de acceso para los componentes individuales crean una nueva **Color** utilizando el valor entrante con una llamada a **Color.FromRgba** o **Color.FromHsla**. este nuevo **Color** valor se establece en el **Color** propiedad en lugar de la **color** campo, lo que significa que el nuevo **Color** valor se somete a procesamiento en el conjunto descriptor de acceso de la **Color** propiedad:

```
clase pública ColorViewModel : INotifyPropertyChanged
{
    Color color;

    evento público PropertyChangedEventHandler PropertyChanged;

    pública doble rojo
    {
        conjunto
        {
            Si (Ronda (color.R)! = valor )
                color = Color .FromRgba ( valor , Color.G, color.B, color.A);
        }
        obtener
        {
            regreso Ronda (color.R);
        }
    }

    pública doble Verde
    {
        conjunto
        {
            Si (Ronda (color.G)! = valor )
                color = Color .FromRgba (color.R, valor , Color.B, color.A);
        }
    }
}
```

```
    obtener
    {
        regreso Ronda (color.G);
    }
}

público doble Azul
{
    conjunto
    {
        Si (Ronda (color.B)!= valor )
            color = Color .FromRgba (color.R, color.G, valor , Color.A);
    }
    obtener
    {
        regreso Ronda (color.B);
    }
}

público doble Alfa
{
    conjunto
    {
        Si (Ronda (color.A)!= valor )
            color = Color .FromRgba (color.R, color.G, color.B, valor );
    }
    obtener
    {
        regreso Ronda (color.A);
    }
}

público doble Matiz
{
    conjunto
    {
        Si (Ronda (color.Hue)!= valor )
            color = Color .FromHsla ( valor , Color.Saturation, color.Luminosity, color.A);
    }
    obtener
    {
        regreso Ronda (color.Hue);
    }
}

público doble Saturación
{
    conjunto
    {
        Si (Ronda (color.Saturation)!= valor )
            color = Color .FromHsla (color.Hue, valor , Color.Luminosity, color.A);
    }
    obtener
    {
```

```
    regreso Ronda (color.Saturation);
}

}

pública doble Luminosidad
{
    conjunto
    {
        Si (Ronda (color.Luminosity) != valor )
            color = Color .FromHsla (color.Hue, color.Saturation, valor , Color.A);
    }
    obtener
    {
        regreso Ronda (color.Luminosity);
    }
}

público Color Color
{
    conjunto
    {
        Color oldColor = color;

        Si (Color != valor )
        {
            color = valor ;
            OnPropertyChanged ( "Color" );
        }

        Si (Color.R != OldColor.R)
            OnPropertyChanged ( "Rojo" );

        Si (Color.G != OldColor.G)
            OnPropertyChanged ( "Verde" );

        Si (Color.B != OldColor.B)
            OnPropertyChanged ( "Azul" );

        Si (Color.A != OldColor.A)
            OnPropertyChanged ( "Alfa" );

        Si (Color.Hue != OldColor.Hue)
            OnPropertyChanged ( "Matiz" );

        Si (Color.Saturation != OldColor.Saturation)
            OnPropertyChanged ( "Saturación" );

        Si (Color.Luminosity != OldColor.Luminosity)
            OnPropertyChanged ( "Luminosidad" );
    }
    obtener
    {
        regreso color;
    }
}
```

```

        }

    protected void OnPropertyChanged (cuerda nombre de la propiedad)
    {
        PropertyChangedEventHandler handler = PropertyChanged;

        Si (Manejador != nulo)
        {
            entrenador de animales( esta , nuevo PropertyChangedEventArgs (nombre de la propiedad));
        }
    }

    doble Redondo( doble valor)
    {
        regreso Dispositivo .OnPlatform (valor, Mates .Round (valor, 3), valor);
    }
}
}

```

los conjunto para el accesor Color propiedad es responsable de los disparos de todos PropertyChanged eventos en función de los cambios en las propiedades.

Note la dependiente del dispositivo Redondo método en la parte inferior de la clase y su uso en el conjunto y obtener accessors de los primeros siete propiedades. Esto se añadió cuando la **MultiColorSliders** muestra en el Capítulo 23, "disparadores y comportamientos", reveló un problema. Android parece ser internamente redondeando los componentes de color, provocando inconsistencias entre las propiedades de ser enviado al **Color.FromRgba** y **Color.FromHsla** métodos y las propiedades de la resultante Color valor, que conducen a infinito conjunto y obtener bucles.

los **HslSliders** crea una instancia del programa ColorViewModel Entre Grid.BindingContext etiquetas de manera que se convierte en el BindingContext para toda la deslizador y Etiqueta elementos dentro de la Cuadrícula:

```

< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    xmlns: kit de herramientas =
        " clr-espacio de nombres: Xamarin.FormsBook.Toolkit; montaje = Xamarin.FormsBook.Toolkit "
    x: Class = " HslSliders.HslSlidersPage "
    SizeChanged = " OnPageSizeChanged " >

    < ContentPage.Padding >
        < OnPlatform x: TypeArguments = " Espesor "
            iOS = " 0, 20, 0, 0 " />
    </ ContentPage.Padding >

    < Cuadrícula x: Nombre = " mainGrid " >
        < Grid.BindingContext >
            < kit de herramientas: ColorViewModel Color = " gris " />
        </ Grid.BindingContext >

        < Grid.Resources >
            < ResourceDictionary >
                < Estilo Tipo de objetivo = " Etiqueta " >
                    < Setter Propiedad = " Tamaño de fuente " Valor = " Grande " />
                </ Estilo >
            </ ResourceDictionary >
        </ Grid.Resources >
    </ Cuadrícula >

```

```

        <Setter Propiedad = "HorizontalTextAlignment" Valor = "Centrar" />
    </Estilo>
</ResourceDictionary>
</Grid.Resources>

<!-- Inicializado para el modo de retrato. -->
<Grid.RowDefinitions>
    <RowDefinition Altura = "*" />
    <RowDefinition Altura = "Auto" />
</Grid.RowDefinitions>

<Grid.ColumnDefinitions>
    <ColumnDefinition Anchura = "*" />
    <ColumnDefinition Anchura = "0" />
</Grid.ColumnDefinitions>

<BoxView Color = "{Binding Color}"
        Grid.Row = "0" Grid.Column = "0" />

<StackLayout x: Nombre = "controlPanelStack"
        Grid.Row = "1" Grid.Column = "0"
        Relleno = "10, 5">

    <StackLayout VerticalOptions = "CenterAndExpand">
        <deslizador Valor = "{Binding Hue}" />
        <Etiqueta Texto = "{Binding Hue, StringFormat = 'Hue = {0: F2}'}" />
    </StackLayout>

    <StackLayout VerticalOptions = "CenterAndExpand">
        <deslizador Valor = "{Binding Saturation}" />
        <Etiqueta Texto = "{Binding Saturation, StringFormat = 'Saturación = {0: F2}'}" />
    </StackLayout>

    <StackLayout VerticalOptions = "CenterAndExpand">
        <deslizador Valor = "{Binding Luminosity}" />
        <Etiqueta Texto = "{Binding Luminosity, StringFormat = 'Luminosidad = {0: F2}'}" />
    </StackLayout>
</StackLayout>
</C cuadrícula>
</Página de contenido>

```

Observe que el **Color** propiedad de **ColorViewModel** cuando se inicializa **ColorViewModel** se crea una instancia. Los enlaces bidireccionales de los deslizadores a continuación recogen los valores resultantes de la La saturación de color, y Luminosidad propiedades.

Si por el contrario desea implementar una pantalla de valores hexadecimales de Rojo verde, y Azul, se puede utilizar el **DoubleToIntConverter** clase demostró en relación con la **GridRgbSliders** programa en el capítulo anterior.

Los HslSliders programa implementa la misma técnica para la conmutación entre modos vertical y horizontal como que **GridRgbSliders** programa.

El archivo de código subyacente se ocupa de la mecánica de este interruptor:

```
público clase parcial HslSlidersPage : Pagina de contenido
{
    público HslSlidersPage ()
    {
        InitializeComponent ();
    }

    vacío OnPageSizeChanged ( objeto remitente, EventArgs args)
    {
        // Modo retrato.
        Si (Anchura <Altura)
        {
            mainGrid.RowDefinitions [1] .height = GridLength .Auto;
            mainGrid.ColumnDefinitions [1] .Width = nuevo GridLength (0, GridUnitType .Absoluto);

            Cuadricula .SetRow (controlPanelStack, 1);
            Cuadricula .SetColumn (controlPanelStack, 0);

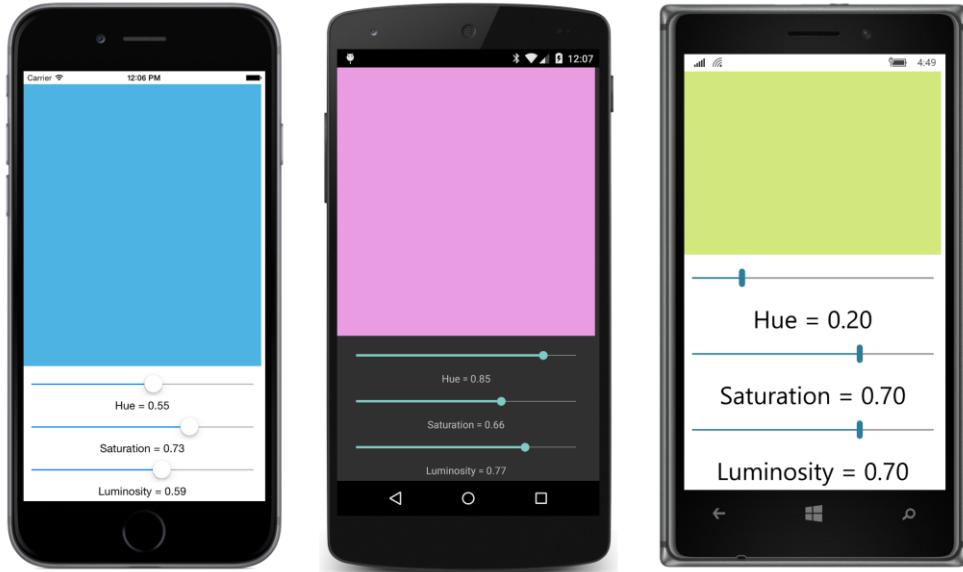
        }
        // Modo paisaje.
        más
        {
            mainGrid.RowDefinitions [1] .height = nuevo GridLength (0, GridUnitType .Absoluto);
            mainGrid.ColumnDefinitions [1] .Width = nuevo GridLength (1, GridUnitType .Estrella);

            Cuadricula .SetRow (controlPanelStack, 0);
            Cuadricula .SetColumn (controlPanelStack, 1);

        }
    }
}
```

Este archivo de código subyacente no es tan bonita como un archivo que se llama simplemente `InitializeComponent`, pero incluso en el contexto de MVVM, el cambio entre los modos vertical y horizontal es un uso legítimo del archivo de código subyacente, ya que se dedica exclusivamente a la interfaz de usuario en lugar de la lógica empresarial subyacente.

Aquí está la **HslSliders** programa en acción:



La racionalización del modelo de vista

Una aplicación típica de INotifyPropertyChanged tiene un campo privado de respaldo para cada propiedad pública definida por la clase, por ejemplo:

```
doble número;
```

También tiene una OnPropertyChanged Método responsable de disparar el PropertyChanged evento:

```
protected void OnPropertyChanged ( cuerda nombre de la propiedad)
{
    PropertyChangedEventHandler handler = PropertyChanged;

    Si (Manejador! = nulo )
    {
        PropertyChanged ( esta , nuevo PropertyChangedEventArgs (nombre de la propiedad));
    }
}
```

Una definición de propiedad típica se parece a esto:

```
pùblico doble Número
{
    conjunto
    {
        Si (Número! = valor )
        {
            nùmero = valor ;
            OnPropertyChanged ( "Número" );
            // Hacer algo con el nuevo valor.
        }
    }
}
```

```

        }
    }
    obtener
    {
        regreso número;
    }
}

```

Un problema potencial consiste en la cadena de texto que se pasa a la `OnPropertyChanged` método. Si se escribe incorrectamente, usted no recibirá ningún tipo de mensaje de error, y sin embargo, fijaciones que implican que la propiedad no va a funcionar. Además, el campo respaldo aparece tres veces dentro de esta única propiedad. Si usted tenía varias propiedades similares y los define a través de operaciones de copiar y pegar, es posible omitir el cambio de nombre de una de las tres apariciones del campo de respaldo, y ese error puede ser muy difícil de localizar.

Puede resolver el primer problema con una característica introducida en C # 5.0. Los `CallerMemberName` atributo le permite reemplazar un argumento método opcional con el nombre del método de llamada o propiedad.

Puede hacer uso de esta función mediante la redefinición de la `OnPropertyChanged` método. Hacer que el argumento opcional mediante la asignación nulo a la misma y que lo precede con el `CallerMemberName` atributo entre corchetes. Usted también necesitará una utilizando la directiva de `System.Runtime.CompilerServices`:

```

protected void OnPropertyChanged ([CallerMemberName] cuerda nombrePropiedad = nulo )
{
    PropertyChangedEventHandler handler = PropertyChanged;

    Si (Manejador! = nulo )
    {
        PropertyChanged (esta , nuevo PropertyChangedEventArgs (nombre de la propiedad));
    }
}

```

Ahora el Número propiedad puede llamar al `OnPropertyChanged` método sin el argumento que indica el nombre de la propiedad. Ese argumento se ajusta automáticamente en el "Número" nombre de la propiedad porque ahí es donde la llamada a `OnPropertyChanged` es originaria:

```

público doble Número
{
    conjunto
    {
        Si (Número! = valor )
        {
            número = valor ;
            OnPropertyChanged ();

            // Hacer algo con el nuevo valor.
        }
    }
    obtener
    {
        regreso número;
    }
}

```

```
}
```

Este enfoque evita un nombre de propiedad de texto mal escrita y también permite a los nombres de propiedades pueden cambiar durante el desarrollo del programa sin tener que preocuparse acerca de cambiar también las cadenas de texto. De hecho, una de las principales razones de que el `CallerMemberName` atributo se inventó era simplificar clases que implementan `INotifyPropertyChanged`.

Sin embargo, esto sólo funciona cuando `OnPropertyChanged` se llama de la propiedad cuyo valor está cambiando. En el anterior `ColorViewModel`, todavía serían necesarios los nombres de propiedad explícita en todas menos una de las llamadas a `OnPropertyChanged`.

Es posible ir aún más lejos de simplificar el conjunto acceso lógica: Tendrá que definir un método genérico, probablemente llamado `SetProperty` o algo similar. Esta `SetProperty` método también se define con la `CallerMemberName` atributo:

```
bool SetProperty <T> ( árbito almacenamiento T, el valor T, [ CallerMemberName ] cuerda nombrePropiedad = nulo )
{
    Si ( Objeto .equals (almacenamiento, valor)
        falso retorno ;

    almacenamiento = valor;
    OnPropertyChanged (propertyName);
    return true ;
}

protected void OnPropertyChanged ([ CallerMemberName ] cuerda nombrePropiedad = nulo )
{
    PropertyChangedEventHandler handler = PropertyChanged;
    Si (Manejador! = nulo )
    {
        PropertyChanged ( esta , nuevo PropertyChangedEventArgs (nombre de la propiedad));
    }
}
```

El primer argumento de `SetProperty` es una referencia al campo de respaldo, y el segundo argumento es el valor que se establece en la propiedad. `SetProperty` automatiza la comprobación y ajuste del campo de respaldo. Note que incluye explícitamente la nombre de la propiedad argumento al llamar `OnPropertyChanged`.

Cambiado. (de lo contrario el nombre de la propiedad argumento se convertiría en la cadena " SetProperty")! devuelve el método cierto si la propiedad fue cambiado. Puede utilizar este valor de retorno para realizar un procesamiento adicional con el nuevo valor.

Ahora el Número la propiedad se ve así:

```
público doble Número
{
    conjunto
    {
        Si ( SetProperty ( árbito número, valor ))
        {
            // Hacer algo con el nuevo valor.
        }
    }
}
```

```

    }
    obtener
    {
        regreso número;
    }
}

```

A pesar de que SetProperty es un método genérico, el C # compilador puede deducir el tipo de los argumentos. Si usted no necesita hacer nada con el nuevo valor en la propiedad conjunto de acceso, puede incluso reducir los dos métodos de acceso a líneas individuales sin ocultar las operaciones de:

```

público doble Número
{
    conjunto {SetProperty ( árbito número, valor ); }
    obtener { regreso número; }
}

```

Es posible que como esta racionalización, tanto que se le quiere poner el SetProperty y OnPropertyChanged métodos en su propia clase y se derivan de esa clase al crear sus propios ViewModels. dicha clase, llamada ViewModelBase, ya está en el **Xamarin.FormsBook.Toolkit** biblioteca:

```

utilizando Sistema;
utilizando System.ComponentModel;
utilizando System.Runtime.CompilerServices;

espacio de nombres Xamarin.FormsBook.Toolkit
{
    clase pública ViewModelBase : INotifyPropertyChanged
    {
        evento público PropertyChangedEventHandler PropertyChanged;

        bool protegida SetProperty <T> ( árbito almacenamiento T, el valor de T,
                                         [ CallerMemberName ] cuerda nombrePropiedad = nulo )
        {
            Si ( Objeto .equals (almacenamiento, valor)
                falso retorno ;

            almacenamiento = valor;
            OnPropertyChanged (propertyName);
            return true ;
        }

        protected void OnPropertyChanged ([ CallerMemberName ] cuerda nombrePropiedad = nulo )
        {
            PropertyChangedEventHandler handler = PropertyChanged;
            Si (Manejador! = nulo )
            {
                PropertyChanged ( esta , nuevo PropertyChangedEventArgs (nombre de la propiedad));
            }
        }
    }
}

```

Esta clase se utiliza en los dos ejemplos de este capítulo.

La interfaz de comandos

Los enlaces de datos son muy poderosos. Los enlaces de datos se conectan propiedades de los elementos visuales en la vista de propiedades de datos en el modelo de vista, y permiten la manipulación directa de los elementos de datos a través de la interfaz de usuario.

Pero no todo es una propiedad. A veces ViewModels exponen pública *métodos* que debe ser llamado desde la vista basada en la interacción de un usuario con un elemento visual. Sin MVVM, lo que probablemente llama a un procedimiento de este tipo a partir de una hecho clic controlador de eventos de un Botón o una aprovechado controlador de eventos de un TapGestureRecognizer. Al considerar estas necesidades, todo el concepto de enlaces de datos y MVVM podría empezar a parecer irremediablemente defectuoso. ¿Cómo puede el archivo de código subyacente de una clase página se desnudó a una InitializeComponent llamar si todavía tiene que hacer llamadas a los métodos de la vista al modelo de vista?

No renunciar a MVVM tan rápido! Xamarin.Forms es compatible con una función que permite a los enlaces de datos para hacer llamadas de método en el modelo de vista directamente desde Botón y TapGestureRecognizer y algunos otros elementos. Este es un protocolo llamado *interfaz de comandos* o el *Interfaz de mando*.

La interfaz de comandos se apoya en ocho clases:

- Botón
- Opción del menú (cubierto en el Capítulo 19, "vistas Collection"), y por tanto también ToolbarItem
- Barra de búsqueda
- TextCell, y por lo tanto también ImageCell (también ser cubierto en el Capítulo 19)
- Vista de la lista (también ser cubierto en el Capítulo 19)
- TapGestureRecognizer

También es posible poner en práctica dominante en sus propias clases personalizadas.

La interfaz de comandos es probable que sea un poco complicado al principio. Vamos a centrarnos en Botón.

Botón define dos formas de código para ser notificados cuando se hace clic en el elemento. El primero es el hecho clic evento. Pero también se puede utilizar la interfaz de comandos del botón como una alternativa a (o además de) la hecho clic evento. Esta interfaz se compone de dos propiedades públicas que Botón define:

- Mando de tipo System.Windows.Input.ICommand.
- CommandParameter de tipo Objeto.

Para apoyar al mando, un modelo de vista debe definir una propiedad pública del tipo Yo ordeno que se conecta entonces a la Mando propiedad de la Botón a través de un conjunto de datos normales de unión.

Me gusta INotifyPropertyChanged, el Yo ordeno interfaz no es una parte de Xamarin.Forms. Se define en el System.Windows.Input espacio de nombres y aplicado en el **System.ObjectModel** montaje, que es uno de los ensamblados de .NET vinculados a una aplicación Xamarin.Forms. Yo ordeno es el

sólo tipo en el System.Windows.Input espacio de nombres que Xamarin.Forms soporta. De hecho es el único tipo de *alguna* System.Windows espacio de nombres con el apoyo de Xamarin.Forms.

¿Es una coincidencia que INotifyPropertyChanged y Yo ordeno están ambos definidos en ensamblados .NET en lugar de Xamarin.Forms? No. Estas interfaces se utilizan a menudo en ViewModels, y algunos desarrolladores ya pueden tener ViewModels desarrollados para uno o más de los entornos basados en XAML de Microsoft. Es más fácil para los desarrolladores incorporar estos ViewModels existentes en Xamarin.Forms si

INotifyPropertyChanged y Yo ordeno se definen en espacios de nombres de .NET estándar y conjuntos en lugar de en Xamarin.Forms.

Los Yo ordeno interfaz define dos métodos y un evento:

```
interfaz pública ICommand
{
    void Ejecutar (objeto arg);

    bool CanExecute (objeto arg);

    evento EventHandler CanExecuteChanged;
}
```

Para implementar de mando, el ViewModel define una o más propiedades de tipo Yo ordeno, lo que significa que la propiedad es un tipo que implementa estos dos métodos y el evento. Una propiedad en el modelo de vista que implementa Yo ordeno a continuación, se pueden unir a la Mando propiedad de una Botón.

Cuando el Botón se hace clic, el Botón dispara sus normales hecho clic evento como de costumbre, sino que también llama la Ejecutar método del objeto unido a su Mando propiedad. El argumento de la Ejecutar método es el objeto establecido en el CommandParameter propiedad de la Botón.

Esa es la técnica básica. Sin embargo, podría ser que ciertas condiciones en el modelo de vista prohíben una Botón haga clic en el momento actual. En ese caso, el Botón debe estar deshabilitado. Este es el propósito de la CanExecute método y la CanExecuteChanged evento en Yo ordeno. Los Botón llamadas CanExecute cuando su Mando propiedad se establece en primer lugar. Si CanExecute devoluciones falso, el Botón se desactiva y no genera Ejecutar llamadas. Los Botón También se instala un controlador para el CanExecuteChanged evento. A partir de entonces, cada vez que el ViewModel dispara el CanExecuteChanged de eventos, las llamadas de botón CanExecute de nuevo para determinar si el botón debe estar habilitado.

A ViewModel que soporta la interfaz de comandos define una o más propiedades de tipo ICOM-Mand e internamente establece esta propiedad a una clase que implementa la Yo ordeno interfaz. ¿Cuál es esta clase, y cómo funciona?

Si estaba implementando el protocolo dominante en uno de los entornos basados en XAML de Microsoft, que estaría escribiendo su propia clase que implementa Yo ordeno, o tal vez utilizando uno que ha encontrado en la web, o uno que se incluye con algunas herramientas MVVM. A veces, estas clases se denominan CommandDelegate o algo similar.

Usted puede utilizar esa misma clase en los ViewModels de sus aplicaciones Xamarin.Forms. Sin embargo, para su comodidad, Xamarin.Forms incluye dos clases que implementan Yo ordeno que se puede utilizar en su lugar. Estas dos clases se denominan simplemente Mando y Comando <T>, donde T es el tipo de los argumentos para Ejecutar y CanExecute.

Si usted es de hecho comparte un modelo de vista entre los ambientes y Xamarin.Forms Microsoft, no se puede utilizar el Mando clases definidas por Xamarin.Forms. Sin embargo, podrás usar algo similar a éstos Mando clases, por lo que la siguiente discusión, sin duda serán de aplicación independientemente.

los Mando clase incluye los dos métodos y eventos de la Yo ordeno interfaz y también define una ChangeCanExecute método. Este método hace que el Mando oponerse a disparar el CanExecute- cambiado acontecimiento, y que las instalaciones resulta ser muy práctico.

Dentro del modelo de vista, es probable que crear un objeto de tipo Mando o Comando <T> para cada propiedad pública en el modelo de vista del tipo de Yo ordeno. los Mando o Comando <T> constructor requiere un método de devolución de llamada en la forma de una Acción objeto que se llama cuando el Botón llama a la Execute método de la Yo ordeno interfaz. los CanExecute método es opcional, pero toma la forma de una Func objeto que devuelve bool.

En muchos casos, las propiedades de tipo Yo ordeno se establecen en el constructor del modelo de vista y no cambian a partir de entonces. Por esa razón, estos Yo ordeno propiedades por lo general no necesitan disparar ApropiadamenteChanged eventos.

ejecuciones método sencillo

Veamos un ejemplo sencillo. Un programa llamado **PowersOfThree** le permite usar dos botones para explorar las diversas potencias de 3. Un botón aumenta el exponente y el otro botón disminuye el exponente.

los PowersViewModel clase se deriva de la ViewModelBase clase en el **Xamarin.FormsBook.Toolkit** biblioteca, pero el propio modelo de vista es en el **PowersOfThree** proyecto de aplicación. No se limita a potencias de 3, pero el constructor requiere un argumento que la clase utiliza como valor de base para el cálculo de la potencia, y que se expone como el Valor base propiedad. Debido a esta propiedad tiene una privada conjunto acceso y no cambia después del constructor llega a la conclusión, la propiedad no se dispara una

PropertyChanged evento.

Otras dos propiedades, el nombre Exponente y Poder, hacer fuego PropertyChanged eventos, pero ambos tienen propiedades también privada conjunto descriptores de acceso. los Exponente propiedad aumenta y disminuye sólo de clics de los botones externos.

Para poner en práctica la respuesta a Botón grifos, la PowersViewModel clase define dos propiedades de tipo Yo ordeno, llamado IncreaseExponentCommand y DecreaseExponentCommand. De nuevo, ambos establecimientos privados conjunto descriptores de acceso. Como se puede ver, el constructor establece estas dos propiedades instanciándolo Mando objetos que hacen referencia a métodos poco privadas inmediatamente después del constructor. Estos dos métodos son llamados pequeños cuando el Ejecutar método de Mando se llama. El modelo de vista utiliza el Mando clase en lugar de Comando <T> porque el programa no hace uso de cualquier

argumento de la Ejecutar métodos:

```
clase PowersViewModel : ViewModelBase
{
    doble exponente, de potencia;

    público PowersViewModel ( doble valor base)
    {
        // Inicializar propiedades.
        BaseValue = baseValue;
        Exponente = 0;

        // propiedades Inicializar ICommand.
        IncreaseExponentCommand = nuevo Mando (ExecuteIncreaseExponent);
        DecreaseExponentCommand = nuevo Mando (ExecuteDecreaseExponent);
    }

    vacio ExecuteIncreaseExponent ()
    {
        Exponente += 1;
    }

    vacio ExecuteDecreaseExponent ()
    {
        Exponente -= 1;
    }

    público doble Valor base { conjunto privado ; obtener ;}

    público doble Exponente
    {
        conjunto privado
        {
            Si ( SetProperty ( árbito exponente, valor ))
            {
                potencia = Mates . Pow (BaseValue, exponente);
            }
        }
        obtener
        {
            regreso exponente;
        }
    }

    público doble Poder
    {
        conjunto privado { SetProperty ( árbito poder, valor );}
        obtener { regreso poder; }
    }

    público Yo ordeno IncreaseExponentCommand { conjunto privado ; obtener ;}

    público Yo ordeno DecreaseExponentCommand { conjunto privado ; obtener ;}
}
```

los `ExecuteIncreaseExponent` y `ExecuteDecreaseExponent` métodos ambos hacen un cambio en el Exponente propiedad (que dispara una `PropertyChanged` evento), y el Exponente vuelve a calcular la propiedad Poder propiedad, que también dispara una `PropertyChanged` evento.

Muy a menudo un modelo de vista creará una instancia de su Mando objetos por los que pasa funciones lambda a la Mando constructor. Este enfoque permite a estos métodos para definir la derecha en el constructor modelo de vista, así:

```
IncreaseExponentCommand = nuevo Mando () =>
{
    Exponente += 1;
};
```

```
DecreaseExponentCommand = nuevo Mando () =>
{
    Exponente -= 1;
};
```

los `PowersOfThreePage` archivo XAML se une la Texto propiedades de tres Etiqueta elementos a la Valor base, Exponente, y Poder propiedades de la `PowersViewModel` clase, y se une la Mando propiedades de las dos Botón elementos a la `IncreaseExponentCommand` y `DecreaseExponentCommand` propiedades del ViewModel.

Note como un argumento de 3 se pasa al constructor de `PowersViewModel` ya que se crea una instancia en el recursos diccionario. Paso de argumentos a los constructores ViewModel es la razón principal de la existencia de la x: Argumentos etiqueta:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
  xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
  xmlns: locales = " CLR-espacio de nombres: PowersOfThree "
  x: Class = " PowersOfThree.PowersOfThreePage " >

  < ContentPage.Resources >
    < ResourceDictionary >
      < locales: PowersViewModel x: Key = " viewmodel " >
        < x: Argumentos >
          < x: Doble > 3 </ x: Doble >
        </ x: Argumentos >
      </ locales: PowersViewModel >
    </ ResourceDictionary >
  </ ContentPage.Resources >

  < StackLayout BindingContext = " {} StaticResource viewmodel " >
    < StackLayout Orientación = " Horizontal "
      Espaciado = " 0 "
      HorizontalOptions = " Centrar "
      VerticalOptions = " CenterAndExpand " >
      < Etiqueta Tamaño de fuente = " Grande "
        Texto = " {Binding BaseValue, StringFormat = '{0}' } " />

      < Etiqueta Tamaño de fuente = " Pequeña "
        Texto = " {Binding Exponente, StringFormat = '{0}' } " />
```

```

< Etiqueta Tamaño de fuente = " Grande "
    Texto = "{Binding potencia, StringFormat = '={0}'}" />
</ StackLayout >

< StackLayout Orientación = " Horizontal "
    VerticalOptions = " CenterAndExpand " >

    < Botón Texto = " Incrementar "
        Mando = "(Binding) IncreaseExponentCommand "
        HorizontalOptions = " CenterAndExpand " />

    < Botón Texto = " Disminución "
        Mando = " { } La unión DecreaseExponentCommand "
        HorizontalOptions = " CenterAndExpand " />

</ StackLayout >
</ StackLayout >
</ Página de contenido >

```

Esto es lo que parece después de varias prensas de un solo botón o la otra:



Una vez más, la sabiduría de la separación de la interfaz de usuario de la lógica de negocio subyacente se revela cuando llega el momento de cambiar la vista. Por ejemplo, supongamos que desea reemplazar los botones con un elemento con una TapGestureRecognizer. Por suerte, TapGestureRecognizer tiene un

Mando propiedad:

```

< StackLayout Orientación = " Horizontal "
    VerticalOptions = " CenterAndExpand " >

    < Marco OutlineColor = " Acento "
        Color de fondo = " Transparente "

```

```

        Relleno = "20, 40"
        HorizontalOptions = "CenterAndExpand" >
    <Frame.GestureRecognizers >
        <TapGestureRecognizer Mando = "{Binding} IncreaseExponentCommand" />
    </Frame.GestureRecognizers >

        <Etiqueta Texto = "Incrementar"
            Tamaño de fuente = "Grande" />
    </Marco >

    <Marco OutlineColor = "Acento"
        Color de fondo = "Transparente"
        Relleno = "20, 40"
        HorizontalOptions = "CenterAndExpand" >
    <Frame.GestureRecognizers >
        <TapGestureRecognizer Mando = "{Binding} DecreaseExponentCommand" />
    </Frame.GestureRecognizers >

        <Etiqueta Texto = "Disminución"
            Tamaño de fuente = "Grande" />
    </Marco >
</StackLayout >

```

Sin tocar el modelo de vista o incluso cambiar el nombre de un controlador de eventos para que se aplique a un grifo en lugar de un botón, el programa funciona de la misma, pero con una mirada diferente:



Una calculadora, casi

Ahora es el momento de hacer un modelo de vista más sofisticado con Yo ordeno objetos que tienen tanto `Execute` y `CanExecute` métodos. El siguiente programa es casi como una calculadora, excepto que sólo se suma una

serie de números. El modelo de vista se llama AdderViewModel, y el programa se llama Una máquina de sumar.

Veamos las capturas de pantalla primeras:



En la parte superior de la página se puede ver una historia de la serie de números que ya han sido introducidos y agregados. Esto es un Etiqueta en un ScrollView, así que puede ser bastante largo.

La suma de esos números se muestra en el Entrada ver por encima del teclado. Normalmente, eso Entrada
vista contiene el número que se está escribiendo, pero después de que se pulsa el signo más grande en la parte derecha del teclado, el Entrada
muestra la suma acumulada y el botón de signo más se desactiva. Que necesita para comenzar a escribir otro número de la suma
acumulada a desaparecer y para el botón con el signo más para estar habilitado. Del mismo modo, el botón de retroceso se activa tan pronto
como se empieza a escribir.

Estas no son las únicas teclas que se pueden desactivar. El punto decimal se desactiva cuando el número que ya está escribiendo tiene
un punto decimal, y todas las teclas numéricas se desactivan cuando el número contiene 16 caracteres. Esto es para evitar el número en el Entrada
llegue a ser demasiado larga para mostrar.

La desactivación de estos botones es el resultado de la aplicación de la CanExecute método en el ICOM-
Mand interfaz.

los AdderViewModel clase es en el Xamarin.FormsBook.Toolkit biblioteca y se deriva de Ver-
ModelBase. Aquí está la parte de la clase con todas las propiedades públicas y sus campos de respaldo:

```
pública clase AdderViewModel : ViewModelBase
{
    cuerda currentEntry = "0";
    cuerda historyString = "";
}
```

```

...
public string CurrentEntry
{
    conjunto privado { SetProperty ( árbito currentEntry, valor ); }
    obtener { regreso currentEntry; }
}

public string HistoryString
{
    conjunto privado { SetProperty ( árbito historyString, valor ); }
    obtener { regreso historyString; }
}

pùblico Yo ordeno ClearCommand { conjunto privado ; obtener ; }

pùblico Yo ordeno ClearEntryCommand { conjunto privado ; obtener ; }

pùblico Yo ordeno BackspaceCommand { conjunto privado ; obtener ; }

pùblico Yo ordeno NumericCommand { conjunto privado ; obtener ; }

pùblico Yo ordeno DecimalPointCommand { conjunto privado ; obtener ; }

pùblico Yo ordeno addCommand { conjunto privado ; obtener ; }

...
}

```

Todos los establecimientos privados conjunto descriptores de acceso. Las dos propiedades de tipo cuerda Sólo se fijan en función internamente en los grifos clave, y las propiedades de tipo Yo ordeno se establecen en el AdderViewModel constructor (que veremos en breve).

Estos ocho propiedades públicas son la única parte de AdderViewModel que el archivo XAML en el Una máquina de sumar proyecto necesita saber sobre. Aquí está el archivo XAML. Contiene una de dos filas y dos columnas principales Cuadrícula para cambiar entre modo vertical y horizontal, y una Etiqueta, de entrada, y 15 Botón elementos, todos los cuales están atados a una de las ocho propiedades públicas del AdderView-Modelo. Observe que el Mando propiedades de los 10 botones numéricos están ligados a la NumericCommand propiedad y que los botones se diferencian por el CommandParameter propiedad. El ajuste de este CommandParameter propiedad se pasa como argumento a la Ejecutar y CanExecute métodos:

```

< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " AddingMachine.AddingMachinePage "
    SizeChanged = " OnPageSizeChanged " >

    < ContentPage.Padding >
        < OnPlatform x: TypeArguments = " Espesor "
            iOS = " 10, 20, 10, 10 "
            Androide = " 10 "
            WinPhone = " 10 " />
    </ ContentPage.Padding >

```

```
< Cuadrícula x: Nombre = " mainGrid " >
    <!-- Inicializado para el modo de retrato. -->
    < Grid.RowDefinitions >
        < RowDefinition Altura = " * " />
        < RowDefinition Altura = " Auto " />
    </ Grid.RowDefinitions >

    < Grid.ColumnDefinitions >
        < ColumnDefinition Anchura = " * " />
        < ColumnDefinition Anchura = " 0 " />
    </ Grid.ColumnDefinitions >

    <!-- visualización de la historia. -->
    < ScrollView Grid.Row = " 0 " Grid.Column = " 0 "
        Relleno = " 5, 0 " >
        < Etiqueta Texto = " \u25b6 La unión HistoryString " />
    </ ScrollView >

    <!-- Teclado. -->
    < Cuadrícula x: Nombre = " keypadGrid " >
        Grid.Row = " 1 " Grid.Column = " 0 "
        Distancia entre filas = " 2 "
        ColumnSpacing = " 2 "
        WidthRequest = " 240 "
        HeightRequest = " 360 "
        VerticalOptions = " Centrar "
        HorizontalOptions = " Centrar " >
        < Grid.Resources >
            < ResourceDictionary >
                < Estilo Tipo de objetivo = " Botón " >
                    < Setter Propiedad = " Tamaño de fuente " Valor = " Grande " />
                    < Setter Propiedad = " Ancho del borde " Valor = " 1 " />
                </ Estilo >
            </ ResourceDictionary >
        </ Grid.Resources >

        < Etiqueta Texto = " \u25b6 La unión CurrentEntry " >
            Grid.Row = " 0 " Grid.Column = " 0 " Grid.ColumnSpan = " 4 "
            Tamaño de fuente = " Grande "
            LineBreakMode = " HeadTruncation "
            VerticalOptions = " Centrar "
            HorizontalTextAlignment = " Fin " />

        < Botón Texto = " do " >
            Grid.Row = " 1 " Grid.Column = " 0 "
            Mando = " {Binding} ClearCommand " />

        < Botón Texto = " CE " >
            Grid.Row = " 1 " Grid.Column = " 1 "
            Mando = " {Binding} ClearEntryCommand " />

        < Botón Texto = " & # X21E6; " >
            Grid.Row = " 1 " Grid.Column = " 2 "
            Mando = " {Binding} BackspaceCommand " />
```

```
< Botón Texto = " + "
  Grid.Row = " 1 " Grid.Column = " 3 " Grid.RowSpan = " 5 "
  Mando = " {} La unión addCommand " />

< Botón Texto = " 7 "
  Grid.Row = " 2 " Grid.Column = " 0 "
  Mando = " {} La unión NumericCommand "
  CommandParameter = " 7 " />

< Botón Texto = " 8 "
  Grid.Row = " 2 " Grid.Column = " 1 "
  Mando = " {} La unión NumericCommand "
  CommandParameter = " 8 " />

< Botón Texto = " 9 "
  Grid.Row = " 2 " Grid.Column = " 2 "
  Mando = " {} La unión NumericCommand "
  CommandParameter = " 9 " />

< Botón Texto = " 4 "
  Grid.Row = " 3 " Grid.Column = " 0 "
  Mando = " {} La unión NumericCommand "
  CommandParameter = " 4 " />

< Botón Texto = " 5 "
  Grid.Row = " 3 " Grid.Column = " 1 "
  Mando = " {} La unión NumericCommand "
  CommandParameter = " 5 " />

< Botón Texto = " 6 "
  Grid.Row = " 3 " Grid.Column = " 2 "
  Mando = " {} La unión NumericCommand "
  CommandParameter = " 6 " />

< Botón Texto = " 1 "
  Grid.Row = " 4 " Grid.Column = " 0 "
  Mando = " {} La unión NumericCommand "
  CommandParameter = " 1 " />

< Botón Texto = " 2 "
  Grid.Row = " 4 " Grid.Column = " 1 "
  Mando = " {} La unión NumericCommand "
  CommandParameter = " 2 " />

< Botón Texto = " 3 "
  Grid.Row = " 4 " Grid.Column = " 2 "
  Mando = " {} La unión NumericCommand "
  CommandParameter = " 3 " />

< Botón Texto = " 0 "
  Grid.Row = " 5 " Grid.Column = " 0 " Grid.ColumnSpan = " 2 "
  Mando = " {} La unión NumericCommand "
  CommandParameter = " 0 " />
```

```

< Botón Texto = " & # X00B7;" 
    Grid.Row = "5" Grid.Column = "2" 
    Mando = " () La unión DecimalPointCommand" />

</ Cuadrícula >
</ Cuadrícula >
</ Página de contenido >

```

Lo que no encontrará en el archivo XAML es una referencia a AdderViewModel. Por razones que veremos en breve, AdderViewModel se crea una instancia de código.

El núcleo de la lógica de la adición-máquina está en el Ejecutar y CanExecute métodos de la seis ICOM-Mand propiedades. Estas propiedades son inicializadas en el AdderViewModel constructor muestra a continuación, y el Ejecutar y CanExecute métodos son todas las funciones lambda.

Cuando sólo hay una función lambda aparece en el Mando constructor, que es la Ejecutar método (como el nombre del parámetro indica), y el Botón siempre está habilitada. Este es el caso de ClearCommand y ClearEntryCommand.

Todo los demás Mando constructores tienen dos funciones lambda. El primero es el Ejecutar método, y el segundo es el CanExecute método. Los CanExecute devuelve el método cierto Si el Botón debe estar habilitado y falso de otra manera.

Todos Yo ordeno propiedades se establecen con la forma no genérico de la Mando a excepción de la clase numericCommand, lo que requiere un argumento a la Ejecutar y CanExecute métodos para identificar qué tecla ha sido aprovechado:

```

público class AdderViewModel : ViewModelBase
{
    ...
    bool isSumDisplayed = falso ;
    doble accumulatedSum = 0;

    público AdderViewModel ()
    {
        ClearCommand = nuevo Mando (
            ejecutar: () =>
            {
                HistoryString = "";
                accumulatedSum = 0;
                CurrentEntry = "0";
                isSumDisplayed = falso ;
                RefreshCanExecutes ();
            });
        ClearEntryCommand = nuevo Mando (
            ejecutar: () =>
            {
                CurrentEntry = "0";
                isSumDisplayed = falso ;
                RefreshCanExecutes ();
            });
    }
}

```

```
BackspaceCommand = nuevo Mando (
    ejecutar: () =>
    {
        CurrentEntry = CurrentEntry.Substring (0, CurrentEntry.Length - 1);

        Si (CurrentEntry.Length == 0)
        {
            CurrentEntry = "0";
        }

        RefreshCanExecutes ();
    },
    CanExecute: () =>
    {
        regreso | IsSumDisplayed && (CurrentEntry.Length> 1 || CurrentEntry [0]! = '0' );
    });
}

NumericCommand = nuevo Mando < cuerda > (
    ejecutar: ( cuerda parámetro) =>
    {
        Si (IsSumDisplayed || CurrentEntry == "0")
            CurrentEntry = parámetro;
        más
            CurrentEntry += parámetro;

        isSumDisplayed = false ;
        RefreshCanExecutes ();
    },
    CanExecute: ( cuerda parámetro) =>
    {
        regreso isSumDisplayed || CurrentEntry.Length <16;
    });
}

DecimalPointCommand = nuevo Mando (
    ejecutar: () =>
    {
        Si (IsSumDisplayed)
            CurrentEntry = "0";
        más
            CurrentEntry += ".";

        isSumDisplayed = false ;
        RefreshCanExecutes ();
    },
    CanExecute: () =>
    {
        regreso isSumDisplayed || (! CurrentEntry.Contains ".");
    });
}

addCommand = nuevo Mando (
    ejecutar: () =>
    {
        doble valor = Doble .Parse (CurrentEntry);
```

```

    HistoryString += value.ToString () + "+";
    accumulatedSum += valor;
    CurrentEntry = accumulatedSum.ToString ();
    isSumDisplayed = cierto ;
    RefreshCanExecutes ();
},
CanExecute: () =>
{
    regreso ! IsSumDisplayed;
});
}

vacio RefreshCanExecutes ()
{
    (( Mando ) BackspaceCommand) .ChangeCanExecute ();
    (( Mando ) NumericCommand) .ChangeCanExecute ();
    (( Mando ) DecimalPointCommand) .ChangeCanExecute ();
    (( Mando ) AddCommand) .ChangeCanExecute ();
}
...
}

```

Todos Ejecutar métodos concluyen llamando a un método llamado RefreshCanExecute siguiendo el constructor. Este método llama al ChangeCanExecute método de cada uno de los cuatro Mando objetos que implementan CanExecute métodos. Eso hace que la llamada al método Mando oponerse a disparar una ChangeCanExecute evento. Cada Botón responde a ese evento al hacer otra llamada a la CanExecute método para determinar si el Botón debe estar habilitado o no.

No es necesario que cada Ejecutar Método para concluir con un llamamiento a los cuatro ChangeCanExecute métodos. Por ejemplo, el ChangeCanExecute método para la DecimalPointCommand no tiene por qué ser llamado cuando el Ejecutar método para NumericCommand ejecuta. Sin embargo, resultó ser más fácil, tanto en términos de la lógica y el código de consolidación a llamar simplemente a todos ellos después de cada grifo llave.

Usted puede ser más cómoda la aplicación de estas Ejecutar y CanExecute métodos como métodos regulares en lugar de las funciones lambda. O puede que sea más cómodo tener un solo comando Mando objeto que se ocupa de todas las teclas. Cada tecla puede tener una identificación CommandParameter cuerdas y se podía distinguir entre ellos con una cambiar y caso declaración.

Hay un montón de maneras de implementar la lógica de mando, pero debe quedar claro que el uso de mando tiende a estructurar el código de una manera flexible e ideal.

Una vez que la lógica de la adición está en su lugar, por qué no añadir un par de botones más para la resta, multiplicación y división?

Bueno, no es tan fácil para mejorar la lógica de aceptar múltiples operaciones en lugar de una sola operación. Si el programa es compatible con múltiples operaciones, a continuación, cuando el usuario teclea una de las teclas de operación, la operación que necesita ser salvado para esperar el siguiente número. Sólo después de que se complete el siguiente número (señalado por la prensa de otra tecla de operación o la tecla igual) es que una operación memorizada aplica.

Un enfoque más fácil sería escribir una calculadora de notación polaca inversa (RPN), donde la operación *siguiente* la entrada de la segunda serie. La simplicidad de la lógica RPN es una de las razones por las cuales las calculadoras RPN apelan a los programadores tanto!

ViewModels y el ciclo de vida de aplicaciones

En un verdadero programa de la calculadora en un dispositivo móvil, una característica importante consiste en el ahorro de todo el estado de la calculadora cuando se termina el programa, y su restauración cuando el programa se pone en marcha de nuevo.

Y una vez más, el concepto del modelo de vista parece romperse.

Claro, es posible escribir algo de código de aplicación que acceda a las propiedades públicas del modelo de vista y las guarda, pero el estado de la calculadora depende de campos privados. Los `isSum-`

Desplegado y `accumulatedSum` campos de `AdderViewModel` son esenciales para la restauración del estado de la calculadora.

Es obvio que el código externo a la `AdderViewModel` no se puede guardar y restaurar la `AdderView-`

Modelo Estado sin el modelo de vista exponer las propiedades más comunes. Sólo hay una clase que sabe lo que es necesario para representar a todo el estado interno de un modelo de vista, y eso es el propio modelo de vista.

La solución es que el modelo de vista para definir métodos públicos que guardar y restaurar su estado interno. Pero debido a que un modelo de vista debe esforzarse por ser independiente de la plataforma, estos métodos no deben usar nada específico para una plataforma en particular. Por ejemplo, no deben tener acceso a los Xamarin.Forms aplicabilidad
ción objeto y luego añadir elementos a (o recuperación de los artículos) de la propiedades Diccionario de ese apli-
cación objeto. Eso es demasiado específico para Xamarin.Forms.

Sin embargo, trabajar con un genérico `IDictionary` objeto en métodos nombrado Guardar Estado y Re-
storeState es posible en *algunas* . entorno de red, y así es como `AdderViewModel` implementa estos métodos:

```
clase pública AdderViewModel : ViewModelBase
{
    ...
    public void Guardar Estado( IDictionary < cuerda , objeto > Diccionario )
    {
        diccionario[ "CurrentEntry" ] = CurrentEntry;
        diccionario[ "HistoryString" ] = HistoryString;
        diccionario[ "IsSumDisplayed" ] = IsSumDisplayed;
        diccionario[ "AccumulatedSum" ] = AccumulatedSum;
    }

    public void RestoreState ( IDictionary < cuerda , objeto > Diccionario )
    {
        CurrentEntry = GetDictionaryEntry ( diccionario, "CurrentEntry" , "0" );
        HistoryString = GetDictionaryEntry ( diccionario, "HistoryString" , " " );
        IsSumDisplayed = GetDictionaryEntry ( diccionario, "IsSumDisplayed" , falso );
    }
}
```

```

    accumulatedSum = GetDictionaryEntry (diccionario, "AccumulatedSum" , 0.0);

    RefreshCanExecutes ();
}

público T GetDictionaryEntry <T> ( IDiccionario < cuerda , objeto > Diccionario,
                                         cuerda clave, T defaultValue)
{
    Si (Dictionary.ContainsKey (clave))
        regreso (T) diccionario [tecla];

    regreso valor por defecto;
}
}

```

El código en **Una máquina de sumar** involucrado en el ahorro y la restauración de este estado se lleva a cabo principalmente en el **Aplicación** clase. Los Aplicación una instancia de la clase **AdderViewModel** y llamadas **RestoreState** utilizando el **propiedades diccionario** de la actual **Solicitud** clase. Ese **AdderViewModel** a continuación, se pasa como argumento a la **AddingMachinePage** constructor:

```

público clase Aplicación : Solicitud
{
    AdderViewModel adderViewModel;

    público App ()
    {
        // Instantie e inicializar modelo de vista de la página.
        adderViewModel = nuevo AdderViewModel ();
        adderViewModel.RestoreState (Current.Properties);
        MainPage = nuevo AddingMachinePage (AdderViewModel);
    }

    protegido override void OnStart ()
    {
        // manipulador cuando su aplicación se inicia.
    }

    protegido override void OnSleep ()
    {
        // manipulador cuando su aplicación duerme.
        adderViewModel.SaveState (Current.Properties);
    }

    protegido override void En resumen()
    {
        // manipulador cuando su aplicación se reanuda.
    }
}

```

Los Aplicación clase también es responsable de llamar **Guardar Estado** en **AdderViewModel** durante el procesamiento de la **OnSleep** método.

Los **AddingMachinePage** constructor sólo tiene que establecer la instancia de **AdderViewModel** al

la página de `BindingContext` propiedad. El archivo de código subyacente también gestiona el cambio entre vertical y horizontal diseños:

```

pública clase parcial AddingMachinePage : Pagina de contenido
{
    pública AddingMachinePage ( AdderViewModel viewmodel)
    {
        InitializeComponent ();

        // Conjunto ViewModel como BindingContext.
        BindingContext = modelo de vista;
    }

    vacío OnPageSizeChanged ( objeto remitente, EventArgs args)
    {
        // Modo retrato.
        Si (Anchura <Altura)
        {
            mainGrid.RowDefinitions [1] .height = GridLength .Auto;
            mainGrid.ColumnDefinitions [1] .Width = nuevo GridLength (0, GridUnitType .Absoluto);

            Cuadrícula .SetRow (keypadGrid, 1);
            Cuadrícula .SetColumn (keypadGrid, 0);
        }
        // Modo paisaje.
        más
        {
            mainGrid.RowDefinitions [1] .height = nuevo GridLength (0, GridUnitType .Absoluto);
            mainGrid.ColumnDefinitions [1] .Width = GridLength .Auto;

            Cuadrícula .SetRow (keypadGrid, 0);
            Cuadrícula .SetColumn (keypadGrid, 1);
        }
    }
}

```

los **Una máquina de sumar** programa muestra una forma de manejar el modelo de vista, pero no es la única manera.

Alternativamente, es posible que Aplicación a instancia del `AdderViewModel` pero definir una propiedad de tipo `AdderViewModel` que el constructor de `AddingMachinePage` puede acceder.

O bien, si desea que la página para tener un control total sobre el modelo de vista, se puede hacer eso también. `AddingMachinePage` puede definir su propio `OnSleep` método que se llama desde el `OnSleep` método en el Aplicación clase, y la clase de página también puede manejar la creación de instancias de `AdderViewModel` y la convocatoria de la `RestoreState` y `Guardar Estado` métodos. Sin embargo, este enfoque podría llegar a ser algo torpe para aplicaciones de múltiples páginas.

En una aplicación de varias páginas, es posible que tenga `ViewModels` separadas para cada página, tal vez se deriva de un modelo de vista con propiedades aplicables a toda la aplicación. En tal caso, tendrá que evitar propiedades con el mismo nombre utilizando las mismas claves de diccionario para guardar el estado de cada modelo de vista. Se puede usar más extensas claves de diccionarios que incluyen el nombre de la clase, por ejemplo, “`AdderViewModel.CurrentEntry`”.

A pesar de la potencia y ventajas de enlace de datos y ViewModels deberían ser evidentes por ahora, estas características realmente florecen cuando se utiliza con los [Xamarin.Forms Vista de la lista](#). Eso depende en el siguiente capítulo.

capítulo 19

vistas Collection

Muchas de las vistas en Xamarin.Forms corresponden a C # .NET de datos y tipos básicos: La deslizador y paso a paso son representaciones visuales de una doble, el Cambiar es un bool, y un Entrada permite al usuario editar el texto expuesto como una cuerda. Sin embargo, esta correspondencia puede también aplicarse a colección tipos en C # y .¿RED?

Colecciones de varias clases han sido siempre esencial en la computación digital. Incluso el más antiguo de los lenguajes de programación de alto nivel apoyan ambas matrices y estructuras. Estas dos colecciones arquetípicas se complementan entre sí: Una matriz es una colección de valores u objetos en general, del mismo tipo, mientras que una estructura es un conjunto de elementos de datos relacionados generalmente de una variedad de tipos.

Para complementar estos tipos de colecciones básicas, .NET añadió varias clases útiles en el System.Collections y System.Collections.Generic espacios de nombres, sobre todo Lista y Lista <T>, que son colecciones expandibles de objetos del mismo tipo. Detrás de estas clases de colección son tres interfaces importantes que te vas a encontrar en este capítulo:

- I Enumerable permite iterar a través de los elementos de una colección.
- I Collection deriva de I Enumerable y añade un recuento de los elementos de la colección.
- I List deriva de I Collection y apoya la indexación, así como añadir y eliminar elementos.

Xamarin.Forms define tres puntos de vista que poseen colecciones de diversos tipos, a veces también permite al usuario seleccionar un elemento de la colección o interactuar con el artículo. Los tres puntos de vista discutidos en este capítulo son:

- Recogedor: Una lista de los elementos de texto que permite al usuario elegir uno. Los Recogedor por lo general mantiene una corta lista de elementos, por lo general no más de una docena más o menos.
- Vista de la lista: Muy a menudo una larga lista de elementos de datos del mismo tipo dictada en un uniforme (o casi uniforme) de manera que se especifica por un árbol visual descrito por un objeto llamado celda.
- TableView: Una colección de células, por lo general de varios tipos, para mostrar los datos o para gestionar la entrada del usuario. UN TableView might take the form of a menu, or a fill-out form, or a collection of application settings.

All three of these views provide built-in scrolling.

At first encounter these three views might seem somewhat similar. The purpose of this chapter is to provide enough examples of how these views are used so that you shouldn't have any difficulty choosing the right tool for the job.

Both Picker and ListView allow selection, but Picker is restricted to strings, while ListView can display any object rendered in whatever way you want. Picker is generally a short list, while ListView can maintain much longer lists.

The relationship between ListView and TableView is potentially confusing because both involve the use of cells, which are derivatives of the Cell class. Cell derives from Element but not VisualElement. A cell is not a visual element itself, but instead provides a description of a visual element. These cells are used by ListView and TableView in two different ways: ListView generally displays a list of objects of the same type, the display of which is specified by a single cell. A TableView is a collection of multiple cells, each of which displays an individual item in a collection of related items.

If you like to equate Xamarin.Forms views with C# and .NET data types, then:

- Picker is a visual representation of an array of string.
- ListView is a more generalized array of objects, often a List<T> collection. The individual items in this collection often implement the INotifyPropertyChanged interface.
- TableView could be a structure, but it is more likely a class, and possibly a class that implements INotifyPropertyChanged, otherwise known as a ViewModel.

Let's begin with the simplest of these three, which is the Picker.

Program options with Picker

Picker is a good choice when you need a view that allows the user to choose one item among a small collection of several items. Picker is implemented in a platform-specific manner and has the limitation that each item is identified solely by a text string.

The Picker and event handling

Here's a program named **PickerDemo** that implements a Picker to allow you to choose a specialized keyboard for an Entry view. In the XAML file, the Entry and the Picker are children of a StackLayout, and the Picker is initialized to contain a list of the various keyboard types supported by the Keyboard class:

```
<ContentPage xmlns = "http://xamarin.com/schemas/2014/forms"
             xmlns:x = "http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class = "PickerDemo.PickerDemoPage" >

    <ContentPage.Padding >
        <OnPlatform x:TypeArguments = "Thickness" 
                    iOS = "0, 20, 0, 0" />
    </ContentPage.Padding >

    <StackLayout Padding = "20"
                Spacing = "50" >
```

```
<Entry x:Name = "entry"
      Placeholder = "Type something, type anything" />

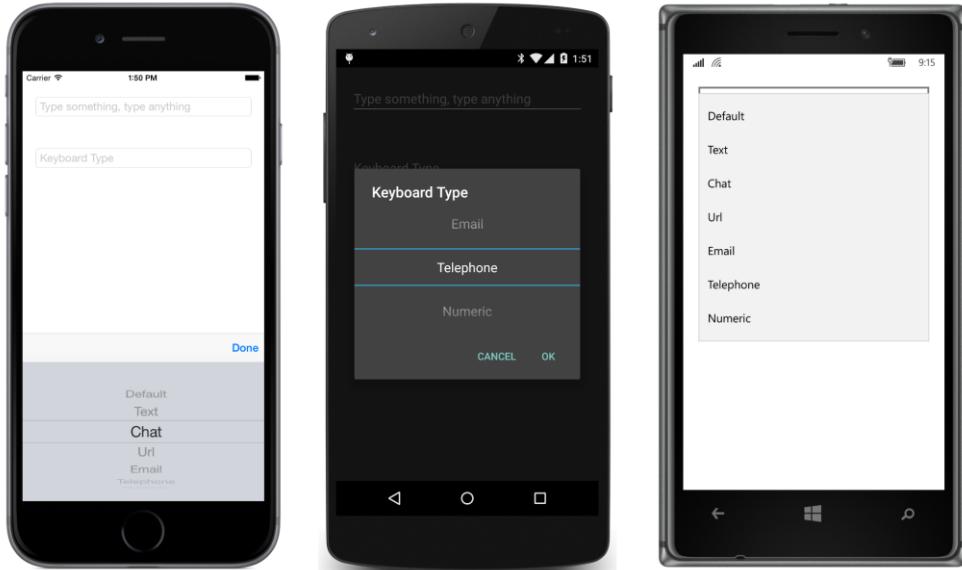
<Picker Title = "Keyboard Type"
        SelectedIndexChanged = "OnPickerSelectedIndexChanged" >
    <Picker.Items>
        <x:String>Default </x:String>
        <x:String>Text </x:String>
        <x:String>Chat </x:String>
        <x:String>Url </x:String>
        <x:String>Email </x:String>
        <x:String>Telephone </x:String>
        <x:String>Numeric </x:String>
    </Picker.Items>
</Picker>
</StackLayout>
</ContentPage>
```

The program sets two properties of Picker: The Title property is a string that identifies the function of the Picker. The Items property is of type `IList<string>`, and generally you initialize it with a list of `x:String` tags in the XAML file. (Picker has no content property attribute, so the explicit `Picker.Items` tags are required.) In code, you can use the Add or Insert method defined by `IList<string>` to put string items into the collection.

Here's what you'll see when you first run the program:



The visual representation of the Picker is quite similar to the Entry but with the Title property displayed. Tapping the Picker invokes a platform-specific scrollable list of items:



When you press **Done** on the iOS screen, or **OK** on the Android screen, or just tap an item on the Windows list, the Picker fires a **SelectedIndexChanged** event. The **SelectedIndex** property of Picker is a zero-based number indicating the particular item the user selected. If no item is selected—which is the case when the Picker is first created and initialized—**SelectedIndex** equals **-1**.

The **PickerDemo** program handles the **SelectedIndexChanged** event in the code-behind file. It obtains the **SelectedIndex** from the Picker, uses that number to index the **Items** collection of the Picker, and then uses reflection to obtain the corresponding **Keyboard** object, which it sets to the **Keyboard** property of the **Entry**:

```
public partial class PickerDemoPage : ContentPage
{
    public PickerDemoPage()
    {
        InitializeComponent();
    }

    void OnPickerSelectedIndexChanged( object sender, EventArgs args )
    {
        if ( entry == null )
            return;

        Picker picker = ( Picker )sender;
        int selectedIndex = picker.SelectedIndex;

        if ( selectedIndex == -1 )
            return;

        string selectedItem = picker.Items[selectedIndex];
        PropertyInfo propertyInfo = typeof( Keyboard ).GetRuntimeProperty(selectedItem);
```

```
        entry.Keyboard = ( Keyboard )propertyInfo.GetValue( null );  
    }  
}
```

At the same time, the interactive Picker display is dismissed, and the Picker now displays the selected item:



On iOS and Android, the selection replaces the Title property, so in a real-life program you might want to provide a simple Label on these two platforms to remind the user of the function of the Picker.

You can initialize the Picker to display a particular item by setting the SelectedIndex property. However, you must set SelectedIndex after filling the Items collection, so you'll probably do it from code or use property-element syntax:

```
< Picker Title = "Keyboard Type" >  
    SelectedIndexChanged = "OnPickerSelectedIndexChanged" >  
    < Picker.Items >  
        < x:String > Default </ x:String >  
        < x:String > Text </ x:String >  
        < x:String > Chat </ x:String >  
        < x:String > Url </ x:String >  
        < x:String > Email </ x:String >  
        < x:String > Telephone </ x:String >  
        < x:String > Numeric </ x:String >  
    </ Picker.Items >  
  
< Picker.SelectedIndex >  
    6  
</ Picker.SelectedIndex >
```

</ Picker >

Data binding the Picker

The `Items` property of `Picker` is not backed by a bindable property; hence, it cannot be the target of a data binding. You cannot bind a collection to a `Picker`. If you need that facility, you'll probably want to use `ListView` instead.

On the other hand, the `SelectedIndex` property of the `Picker` is backed by a `BindableProperty` and has a default binding mode of `TwoWay`. This seems to suggest that you can use `SelectedIndex` in a data binding, and that is true. However, an integer index is usually not what you want in a data binding.

Even if `Picker` had a `SelectedItem` property that provided the actual item rather than the index of the item, that wouldn't be optimum either. This hypothetical `SelectedItem` property would be of type `string`, and usually that's not very useful in data bindings either.

After contemplating this problem—and perhaps being exposed to the `ListView` coming up next—you might try to create a class named `BindablePicker` that derives from `Picker`. Such a class could have an `ObjectItems` property of type `IList<object>` and a `SelectedItem` property of type `object`. However, without any additional information, this `BindablePicker` class would be forced to convert each object in the collection to a string for the underlying `Picker`, and the only generalized way to convert an object to a string is with the object's `ToString` method. Perhaps the string obtained from `ToString` is useful; perhaps not. (You'll see shortly how the `ListView` solves this problem in a very flexible manner.)

Perhaps a better solution for data binding a `Picker` is a value converter that converts between the `SelectedIndex` property of the `Picker` and an object corresponding to each text string in the `Items` collection. To accomplish this conversion, the value converter can maintain its own collection of objects that correspond to the strings displayed by the `Picker`. This means that you'll have two lists associated with the `Picker`—one list of strings displayed by the `Picker` and another list of objects associated with these strings. These two lists must be in exact correspondence, of course, but if the two lists are defined close to each other in the XAML file, there shouldn't be much confusion, and the scheme will have the advantage of being very flexible.

Such a value converter might be called `IndexToObjectConverter`.

Or maybe not. In the general case, you'll want the `SelectedIndex` property of the `Picker` to be the *target* of the data binding. If `SelectedIndex` is the data-binding target, then the `Picker` can be used with a `ViewModel` as the data-binding source. For that reason, the value converter is better named `ObjectToIndexConverter`. Here's the class in the `Xamarin.FormsBook.Toolkit` library:

```
using System;
using System.Collections.Generic;
using System.Globalization;
using Xamarin.Forms;
```

```

namespace Xamarin.FormsBook.Toolkit
{
    [ContentProperty ("Items")]
    public class ObjectToIndexConverter <T> : IValueConverter
    {
        public IList <T> Items { set; get; }

        public ObjectToIndexConverter()
        {
            Items = new List <T>();
        }

        public object Convert( object value, Type targetType,
                               object parameter, CultureInfo culture)
        {
            if (value == null || !(value is T) || Items == null )
                return -1;

            return Items.IndexOf((T)value);
        }

        public object ConvertBack( object value, Type targetType,
                               object parameter, CultureInfo culture)
        {
            int index = ( int )value;

            if (index < 0 || Items == null || index >= Items.Count)
                return null ;

            return Items[index];
        }
    }
}

```

This is a generic class, and it defines a public `Items` property of type `IList<T>`, which is also defined as the content property of the converter. The `Convert` method assumes that the `value` parameter is an object of type `T` and returns the index of that object within the collection. The `ConvertBack` method assumes that the `value` parameter is an index into the `Items` collection and returns that object.

The **PickerBinding** program uses the `ObjectToIndexConverter` to define a binding that allows a Picker to be used for selecting a font size for a Label. The Picker is the data-binding target and the `FontSize` property of the Label is the source. The Binding object is instantiated in element tags to allow the `ObjectToIndexConverter` to be instantiated and initialized locally and provide an easy visual confirmation that the two lists correspond to the same values:

```

<ContentPage xmlns = " http://xamarin.com/schemas/2014/forms "
              xmlns:x = " http://schemas.microsoft.com/winfx/2009/xaml "
              xmlns:toolkit =
                  " clr-namespace:Xamarin.FormsBook.Toolkit;assembly=Xamarin.FormsBook.Toolkit "
              x:Class = " PickerBinding.PickerBindingPage " >

```

```
< ContentPage.Padding >
    < OnPlatform x:TypeArguments = "Thickness" 
        iOS = "0, 20, 0, 0" />
</ ContentPage.Padding >

< StackLayout Padding = "20" 
    Spacing = "50" >

    < Label x:Name = "label" 
        Text = "Sample Text" 
        FontSize = "16" />

    < Picker Title = "Font Size" >
        < Picker.Items >
            < x:String > Font Size = 8 </ x:String >
            < x:String > Font Size = 10 </ x:String >
            < x:String > Font Size = 12 </ x:String >
            < x:String > Font Size = 14 </ x:String >
            < x:String > Font Size = 16 </ x:String >
            < x:String > Font Size = 20 </ x:String >
            < x:String > Font Size = 24 </ x:String >
            < x:String > Font Size = 30 </ x:String >
        </ Picker.Items >

        < Picker.SelectedIndex >
            < Binding Source = "{x:Reference label}" 
                Path = "FontSize" >
                < Binding.Converter >
                    < toolkit:ObjectToIndexConverter x:TypeArguments = "x:Double" >
                        < x:Double > 8 </ x:Double >
                        < x:Double > 10 </ x:Double >
                        < x:Double > 12 </ x:Double >
                        < x:Double > 14 </ x:Double >
                        < x:Double > 16 </ x:Double >
                        < x:Double > 20 </ x:Double >
                        < x:Double > 24 </ x:Double >
                        < x:Double > 30 </ x:Double >
                    </ toolkit:ObjectToIndexConverter >
                </ Binding.Converter >
            </ Binding >
        </ Picker.SelectedIndex >
    </ Picker >
</ StackLayout >
</ ContentPage >
```

By maintaining separate lists of strings and objects, you can make the strings whatever you want. In this case, they include some text to indicate what the number actually means. The Label itself is initialized with a FontSize setting of 16, and the binding picks up that value to display the corresponding string in the Picker when the program first starts up:



The implementations of Picker on these three platforms should make it obvious that you don't want to use the Picker for more than (say) a dozen items. It's convenient and easy to use, but for lots of items, you want a view made for the job—a view that is designed to display objects not just as simple text strings but with whatever visuals you want.

Rendering data with ListView

Let's move to **ListView**, which is the primary view for displaying collections of items, usually of the same type. The ListView always displays the items in a vertical list and implements scrolling if necessary.

ListView is the only class that derives from `ItemsView<T>`, but from that class it inherits its most important property: `ItemsSource` of type `IEnumerable`. To this property a program sets an enumerable collection of data, and it can be any type of data. For that reason, ListView is one of the backbones of the View part of the Model-View-ViewModel architectural pattern.

ListView also supports single-item selection. The ListView highlights the selected item and makes it available as the `SelectedItem` property. Notice that this property is named `SelectedItem` rather than `SelectedIndex`. The property is of type `object`. If no item is currently selected in the ListView, the property is null. ListView fires an `ItemSelected` event when the selected item changes, but often you'll be using data binding in connection with the `SelectedItem` property.

ListView defines more properties by far than any other single view in Xamarin.Forms. The discussion in this chapter begins with the most important properties and then progressively covers the more obscure and less common properties.

Collections and selections

The **ListViewList** program defines a ListView that displays 17 Xamarin.Forms Color values. The XAML file instantiates the ListView but leaves the initialization to the code-behind file:

```
<ContentPage xmlns = "http://xamarin.com/schemas/2014/forms"
    xmlns:x = "http://schemas.microsoft.com/winfx/2009/xaml"
    x:Class = "ListViewList.ListViewListPage" >

    <ContentPage.Padding>
        <OnPlatform x:TypeArguments = "Thickness" 
            iOS = "10, 20, 10, 0"
            Android = "10, 0"
            WinPhone = "10, 0" />
    </ContentPage.Padding>

    <ListView x:Name = "listView" />

</ContentPage>
```

The bulk of this XAML file is devoted to setting a Padding so that the ListView doesn't extend to the left and right edges of the screen. In some cases, you might want to set an explicit WidthRequest for the ListView based on the width of the widest item that you anticipate.

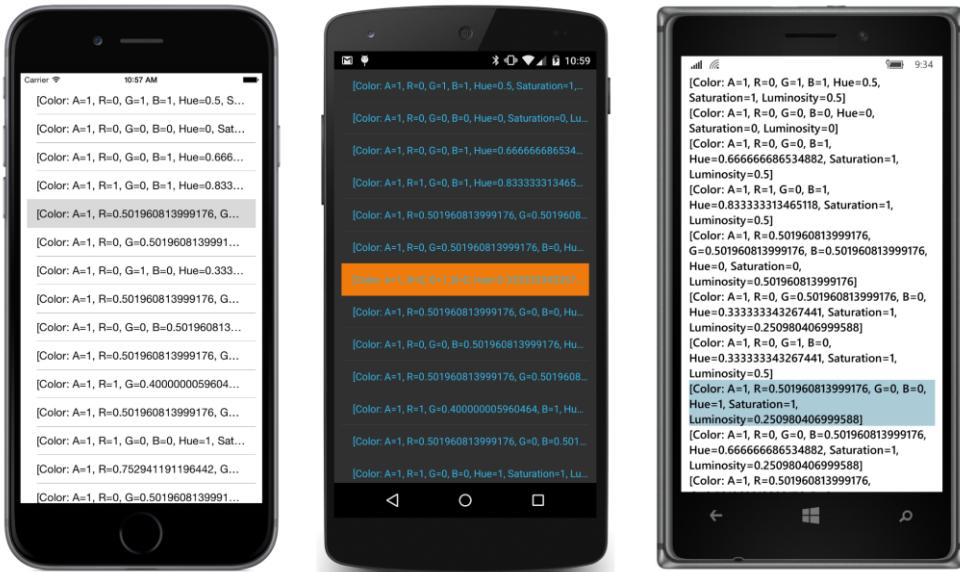
The ItemsSource property of ListView is of type IEnumerable, an interface implemented by arrays and the List class, but the property is null by default. Unlike the Picker, the ListView does not provide its own collection object. That's your responsibility. The code-behind file of **ListViewList**

sets the ItemsSource property to an instance of List<Color> that is initialized with Color values:

```
public partial class ListViewListPage : ContentPage
{
    public ListViewListPage()
    {
        InitializeComponent();

        listView.ItemsSource = new List < Color >
        {
            Color.Aqua, Color.Black, Color.Blue, Color.Fuchsia,
            Color.Gray, Color.Green, Color.Lime, Color.Maroon,
            Color.Navy, Color.Olive, Color.Pink, Color.Purple,
            Color.Red, Color.Silver, Color.Teal, Color.White, Color.Yellow
        };
    }
}
```

When you run this program, you'll discover that you can scroll through the items and select one item by tapping it. These screenshots show how the selected item is highlighted on the three platforms:



Tapping an item also causes the ListView to fire both an ItemTapped and an ItemSelected event. If you tap the same item again, the ItemTapped event is fired again but not the ItemSelected event. The ItemSelected event is fired only if the SelectedItem property changes.

Of course, the items themselves aren't very attractive. By default, the ListView displays each item by calling the item's ToString method, and that's what you see in this ListView. But do not fret: Much of the discussion about the ListView in this chapter focuses on making the items appear exactly how you'd like!

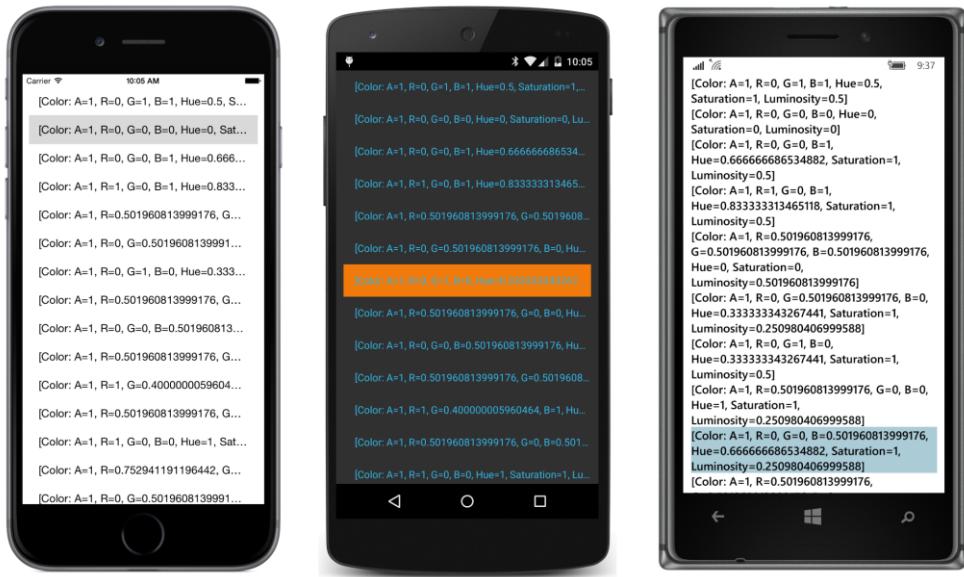
The row separator

Look closely at the iOS and Android displays and you'll see a thin line separating the rows. You can suppress the display of that row by setting the SeparatorVisibility property to the enumeration member SeparatorVisibility.None. The default is SeparatorVisibility.Default, which means that a separator line is displayed on the iOS and Android screens but not Windows Phone.

For performance reasons, you should set the SeparatorVisibility property before adding items to the ListView. You can try this in the ListViewList program by setting the property in the XAML file:

```
<ListView x:Name = "listView"
          SeparatorVisibility = "None" />
```

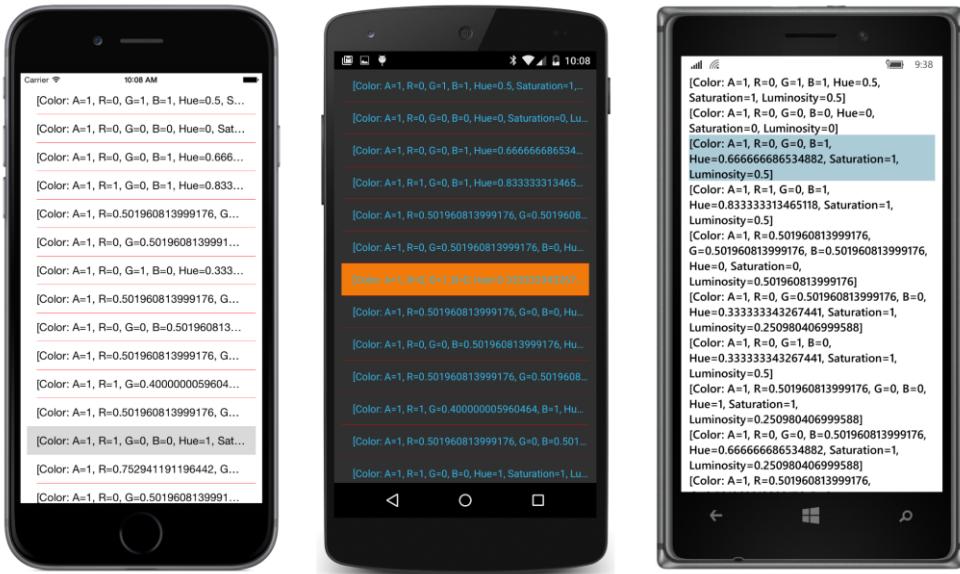
Here's how it looks:



You can also set the separator line to a different color with the `SeparatorColor` property; for example:

```
<ListView x:Name = "listView"
          SeparatorColor = "Red" />
```

Now it shows up in red:



The line is rendered in a platform-specific manner. On iOS, that means it doesn't extend fully to the left edge of the ListView, and on the Windows platforms, that means that there's no separator line at all.

Data binding the selected item

One approach to working with the selected item involves handling the `ItemSelected` event of the `ListView` in the code-behind file and using the `SelectedItem` property to obtain the new selected item. (An example is shown later in this chapter.) But in many cases you'll want to use a data binding with the `SelectedItem` property. The `ListViewArray` program defines a data binding between the

`SelectedItem` property of the `ListView` with the `Color` property of a `BoxView`:

```
<ContentPage xmlns = "http://xamarin.com/schemas/2014/forms"
             xmlns:x = "http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class = "ListViewArray.ListViewArrayPage" >

    <ContentPage.Padding >
        <OnPlatform x:TypeArguments = "Thickness" >
            iOS = "10, 20, 10, 0"
            Android = "10, 0"
            WinPhone = "10, 0" </>
        </ContentPage.Padding >

    <StackLayout >
        <ListView x:Name = "listView"
                  SelectedItem = "{Binding Source={x:Reference boxView},
                                         Path=Color,
                                         Mode=TwoWay}" >
            <ListView.ItemsSource >
                <x:Array Type = "{x:Type Color}" >
```

```

<x:Static Member = "Color.Aqua" />
<x:Static Member = "Color.Black" />
<x:Static Member = "Color.Blue" />
<x:Static Member = "Color.Fuchsia" />
<x:Static Member = "Color.Gray" />
<x:Static Member = "Color.Green" />
<x:Static Member = "Color.Lime" />
<x:Static Member = "Color.Maroon" />
<Color>Navy</Color>
<Color>Olive</Color>
<Color>Pink</Color>
<Color>Purple</Color>
<Color>Red</Color>
<Color>Silver</Color>
<Color>Teal</Color>
<Color>White</Color>
<Color>Yellow</Color>
</x:Array>
</ListView.ItemsSource>
</ListView>

<BoxView x:Name = "boxView"
         Color = "Lime"
         HeightRequest = "100" />

</StackLayout>
</ContentPage>

```

This XAML file sets the `ItemsSource` property of the `ListView` directly from an array of items.

`ItemsSource` is *not* the content property of `ListView` (in fact, `ListView` has no content property at all), so you'll need explicit `ListView.ItemsSource` tags. The `x:Array` element requires a `Type` attribute indicating the type of the items in the array. For the sake of variety, two different approaches of specifying a `Color` value are shown. You can use anything that results in a value of type `Color`.

The `ItemsSource` property of `ListView` is always populated with objects rather than visual elements. For example, if you want to display strings in the `ListView`, use string objects from code or `x:String` elements in the XAML file. Do not fill the `ItemsSource` collection with `Label` elements!

The `ListView` is scrollable, and normally when a scrollable view is a child of a `StackLayout`, a `VerticalOptions` setting of `FillAndExpand` is required. However, the `ListView` itself sets its `HorizontalOptions` and `VerticalOptions` properties to `FillAndExpand`.

The data binding targets the `SelectedItem` property of the `ListView` from the `Color` property of the `BoxView`. You might be more inclined to reverse the source and target property of that binding like this:

```

<BoxView x:Name = "boxView"
         Color = "{Binding Source={x:Reference listView},
                           Path=SelectedItem}"
         HeightRequest = "100" />

```

However, the `SelectedItem` property of the `ListView` is null by default, which indicates that nothing is selected, and the binding will fail with a `NullReferenceException`. To make the binding on the `BoxView` work, you would need to initialize the `SelectedItem` property of the `ListView` after the items have been added:

```
< ListView x:Name = " listView " >
    < ListView.ItemsSource >
        < x:Array Type = " {x:Type Color} " >
            ...
        </ x:Array >
    </ ListView.ItemsSource >

    < ListView.SelectedItem >
        < Color > Lime </ Color >
    </ ListView.SelectedItem >
</ ListView >
```

A better approach—and one that you'll be using in conjunction with MVVM—is to set the binding on the `SelectedItem` property of the `ListView`. The default binding mode for `SelectedItem` is `OneWayToSource`, which means that the following binding sets the `Color` of the `BoxView` to whatever item is selected in the `ListView`:

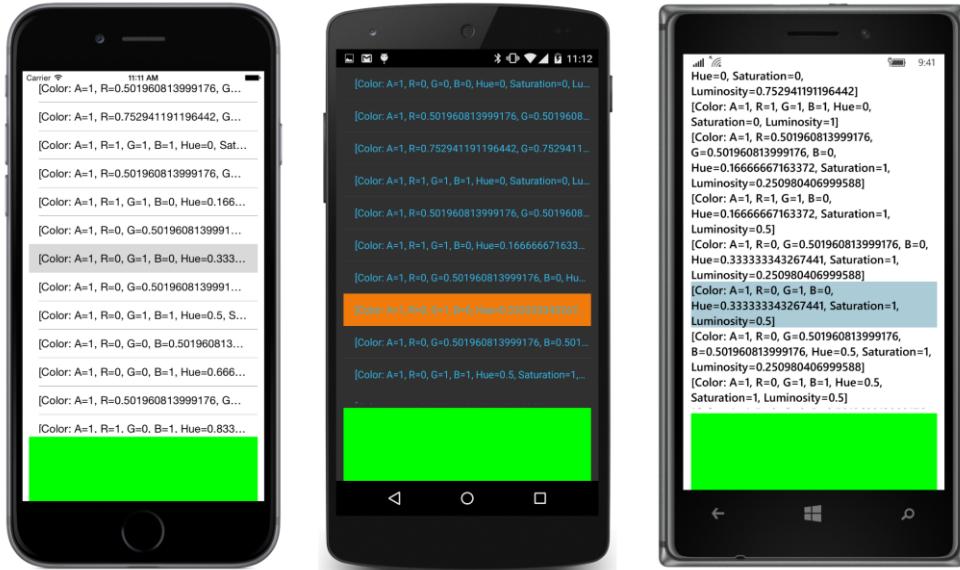
```
< ListView x:Name = " listView "
    SelectedItem = " {Binding Source={x:Reference boxView},
        Path=Color} " >
    ...
</ ListView >
```

However, if you also want to initialize the `SelectedItem` property from the binding source, use a `TwoWay` binding as shown in the XAML file in the `ListViewArray` program:

```
< StackLayout >
    < ListView x:Name = " listView "
        SelectedItem = " {Binding Source={x:Reference boxView},
            Path=Color,
            Mode=TwoWay} " >
    ...
</ ListView >

    < BoxView x:Name = " boxView "
        Color = " Lime "
        HeightRequest = " 100 " />
</ StackLayout >
```

You'll see that the "Lime" entry in the `ListView` is selected when the program starts up:



Actually, it's hard to tell whether that really is the "Lime" entry without examining the RGB values. Although the Color structure defines a bunch of static fields with color names, Color values themselves are not identifiable by name. When the data binding sets a Lime color value to the SelectedItem

property of the ListView, the ListView probably finds a match among its contents using the Equals method of the Color structure, which compares the components of the two colors.

The improvement of the ListView display is certainly a high priority!

If you examine the ListViewArray screen very closely, you'll discover that the Color items are not displayed in the same order in which they are defined in the array. The ListViewArray program has another purpose: to demonstrate that the ListView does not make a copy of the collection set to its ItemsSource property. Instead, it uses that collection object directly as a source of the items. In the code-behind file, after the InitializeComponent call returns, the constructor of ListViewArray-

Page performs an in-place array sort to order the items by Hue:

```
public partial class ListViewArrayPage : ContentPage
{
    public ListViewArrayPage()
    {
        InitializeComponent();

        Array .Sort< Color >(( Color [])listView.ItemsSource,
            ( Color color1, Color color2 ) =>
        {
            if (color1.Hue == color2.Hue)
                return Math .Sign(color1.Luminosity - color2.Luminosity);

            return Math .Sign(color1.Hue - color2.Hue);
        });
    }
}
```

```
    }  
}
```

This sorting occurs after the `ItemsSource` property is set, which occurs when the XAML is parsed by the `InitializeComponent` call, but before the `ListView` actually displays its contents during the layout process.

This code implies that you can change the collection used by the `ListView` dynamically. However, if you want a `ListView` to change its display when the collection changes, the `ListView` must somehow be notified that changes have occurred in the collection that is referenced by its `ItemsSource` property.

Let's examine this problem in more detail.

The `ObservableCollection` difference

The `ItemsSource` property of `ListView` is of type `IEnumerable`. Arrays implement the `IEnumerable` interface, and so do the `List` and `List<T>` classes. The `List` and `List<T>` collections are particularly popular for `ListView` because these classes can dynamically reallocate memory to accommodate a collection of almost any size.

You've seen that a collection can be modified after it's been assigned to the `ItemsSource` property of a `ListView`. It should be possible to add items or remove items from the collection referenced by `ItemsSource`, and for the `ListView` to update itself to reflect those changes.

Let's try it. This `ListViewLogger` program instantiates a `ListView` in its XAML file:

```
<ContentPage xmlns = "http://xamarin.com/schemas/2014/forms"  
    xmlns:x = "http://schemas.microsoft.com/winfx/2009/xaml"  
    x:Class = "ListViewLogger.ListViewLoggerPage">  
  
    <ContentPage.Padding>  
        <OnPlatform x:TypeArguments = "Thickness"  
            iOS = "10, 20, 10, 0"  
            Android = "10, 0"  
            WinPhone = "10, 0" />  
    </ContentPage.Padding>  
  
    <ListView x:Name = "listView" />  
</ContentPage>
```

The code-behind file sets the `ItemsSource` property of the `ListView` to a `List<DateTime>` object and adds a `DateTime` value to this collection every second:

```
public partial class ListViewLoggerPage : ContentPage  
{  
    public ListViewLoggerPage()  
    {  
        InitializeComponent();  
  
        List<DateTime> list = new List<DateTime>();
```

```
listView.ItemsSource = list;

Device.StartTimer(TimeSpan.FromSeconds(1), () =>
{
    list.Add(DateTime.Now);
    return true;
});
}
}
```

When you first run this program, it will seem as if nothing is happening. But if you turn the phone or emulator sideways, all the items that have been added to the collection since the program started will be displayed. But you won't see any more until you turn the phone's orientation again.

What's happening? When the ListView needs to redraw itself—which is the case when you change the orientation of the phone or emulator—it will use the current `IEnumerable` collection. (This is how the `ListViewArray` program displayed the sorted array. The array was sorted before the ListView displayed itself for the first time.)

However, if the ListView does not need to redraw itself, there is no way for the ListView to know when an item has been added to or removed from the collection. This is not the fault of ListView. It's really the fault of the List class. The List and `List<T>` classes don't implement a notification mechanism that signals the ListView when the collection has changed.

To persuade a ListView to keep its display updated with newly added data, we need a class very much like `List<T>`, but which includes a notification mechanism.

We need a class exactly like `ObservableCollection`.

`ObservableCollection` is a .NET class. It is defined in the `System.Collections.ObjectModel` namespace, and it implements an interface called `INotifyCollectionChanged`, which is defined in the `System.Collections.Specialized` namespace. In implementing this interface, an `ObservableCollection` fires a `CollectionChanged` event whenever items are added to or removed from the collection, or when items are replaced or reordered.

How does ListView know that an `ObservableCollection` object is set to its `ItemsSource` property? When the `ItemsSource` property is set, the ListView checks whether the object set to the property implements `INotifyCollectionChanged`. If so, the ListView attaches a `CollectionChanged` handler to the collection to be notified of changes. Whenever the collection changes, the ListView updates itself.

The `ObservableLogger` program is identical to the `ListViewLogger` program except that it uses an `ObservableCollection<DateTime>` rather than a `List<DateTime>` to maintain the collection:

```
public partial class ObservableLoggerPage : ContentPage
{
    public ObservableLoggerPage()
    {
        InitializeComponent();
    }
}
```

```
ObservableCollection < DateTime > list = new ObservableCollection < DateTime >();
listView.ItemsSource = list;

Device.StartTimer( TimeSpan.FromSeconds(1), () =>
{
    list.Add( DateTime.Now);
    return true ;
});
}
```

Now the ListView updates itself every second.

Of course, not every application needs this facility, and ObservableCollection is overkill for those that don't. But it's an essential part of versatile ListView usage.

Sometimes you'll be working with a collection of data items, and the collection itself does not change dynamically—in other words, it always contains the same objects—but properties of the individual items change. Can the ListView respond to changes of that sort?

Yes it can, and you'll see an example later in this chapter. Enabling a ListView to respond to property changes in the individual items does not require ObservableCollection or INotifyCollectionChanged. But the data items must implement INotifyPropertyChanged, and the ListView must display the items using an object called a *cell*.

Templates and cells

The purpose of ListView is to display data. In the real world, data is everywhere, and we are compelled to write computer programs to deal with this data. In programming tutorials such as this book, however, data is harder to come by. So let's invent a little bit of data to explore ListView in more depth, and if the data turns out to be otherwise useful, so much the better!

As you know, the colors supported by the Xamarin.Forms Color structure are based on the 16 colors defined in the HTML 4.01 standard. Another popular collection of colors is defined in the Cascading Style Sheets (CSS) 3.0 standard. That collection contains 147 named colors (seven of which are duplicates for variant spellings) that were originally derived from color names in the X11 windowing system but converted to camel case.

The NamedColor class included in the **Xamarin.FormsBook.Toolkit** library lets your Xamarin.Forms program get access to those 147 colors. The bulk of NamedColor is the definition of 147 public static read-only fields of type Color. Only a few are shown in an abbreviated list toward the end of the class:

```
public class NamedColor
{
    // Instance members.
    private NamedColor()
    {
```

```
}

public string Name { private set; get; }

public string FriendlyName { private set; get; }

public Color Color { private set; get; }

public string RgbDisplay { private set; get; }

// Static members.

static NamedColor()
{
    List<NamedColor> all = new List<NamedColor>();
    StringBuilder stringBuilder = new StringBuilder();

    // Loop through the public static fields of type Color.
    foreach (FieldInfo fieldInfo in typeof(NamedColor).GetRuntimeFields())
    {
        if (fieldInfo.IsPublic &&
            fieldInfo.IsStatic &&
            fieldInfo.FieldType == typeof(Color))
        {
            // Convert the name to a friendly name.
            string name = fieldInfo.Name;
            stringBuilder.Clear();
            int index = 0;

            foreach (char ch in name)
            {
                if (index != 0 && Char.IsUpper(ch))
                {
                    stringBuilder.Append(' ');
                }
                stringBuilder.Append(ch);
                index++;
            }

            // Instantiate a NamedColor object.
            Color color = (Color)fieldInfo.GetValue(null);

            NamedColor namedColor = new NamedColor
            {
                Name = name,
                FriendlyName = stringBuilder.ToString(),
                Color = color,
                RgbDisplay = String.Format("{0:X2}-{1:X2}-{2:X2}",
                                           (int)(255 * color.R),
                                           (int)(255 * color.G),
                                           (int)(255 * color.B))
            };
        }
    }

    // Add it to the collection.
    all.Add(namedColor);
}
```

```

        }
    }

    all.TrimExcess();
    All = all;
}

public static IList<NamedColor> All { private set; get; }

// Color names and definitions from http://www.w3.org/TR/css3-color/
// (but with color names converted to camel case).
public static readonly Color AliceBlue = Color.FromRgb(240, 248, 255);
public static readonly Color AntiqueWhite = Color.FromRgb(250, 235, 215);
public static readonly Color Aqua = Color.FromRgb(0, 255, 255);
...
public static readonly Color WhiteSmoke = Color.FromRgb(245, 245, 245);
public static readonly Color Yellow = Color.FromRgb(255, 255, 0);
public static readonly Color YellowGreen = Color.FromRgb(154, 205, 50);
}

```

If your application has a reference to **Xamarin.FormsBook.Toolkit** and a using directive for the `Xamarin.FormsBook.Toolkit` namespace, you can use these fields just like the static fields in the `Color` structure. For example:

```

BoxView boxView = new BoxView
{
    Color = NamedColor.Chocolate
};

```

You can also use them in XAML without too much more difficulty. If you have an XML namespace declaration for the **Xamarin.FormsBook.Toolkit** assembly, you can reference `NamedColor` in an `x:Static` markup extension:

```
<BoxView Color = "{x:Static toolkit:NamedColor.CornflowerBlue}" />
```

But that's not all: In its static constructor, `NamedColor` uses reflection to create 147 instances of the `NamedColor` class that it stores in a list that is publicly available from the static `All` property. Each instance of the `NamedColor` class has a `Name` property, a `Color` property of type `Color`, a `FriendlyName` property that is the same as the `Name` except with some spaces inserted, and an `RgbDisplay` property that formats the hexadecimal color values.

The `NamedColor` class does not derive from `BindableObject` and does not implement `INotifyPropertyChanged`. Regardless, you can use this class as a binding source. That's because these properties remain constant after each `NamedColor` object is instantiated. Only if these properties later changed would the class need to implement `INotifyPropertyChanged` to serve as a successful binding source.

The `NamedColor.All` property is defined to be of type `IList<NamedColor>`, so we can set it to the `ItemsSource` property of a `ListView`. This is demonstrated by the `NaiveNamedColorList` program:

```
<ContentPage xmlns = "http://xamarin.com/schemas/2014/forms"
```

```

xmlns:x = "http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:toolkit =
    "clr-namespace:Xamarin.FormsBook.Toolkit;assembly=Xamarin.FormsBook.Toolkit"
x:Class = "NaiveNamedColorList.NaiveNamedColorListPage" >

<ContentPage.Padding >
    <OnPlatform x:TypeArguments = "Thickness" >
        iOS = "10, 20, 10, 0"
        Android = "10, 0"
        WinPhone = "10, 0" />
    </ContentPage.Padding >

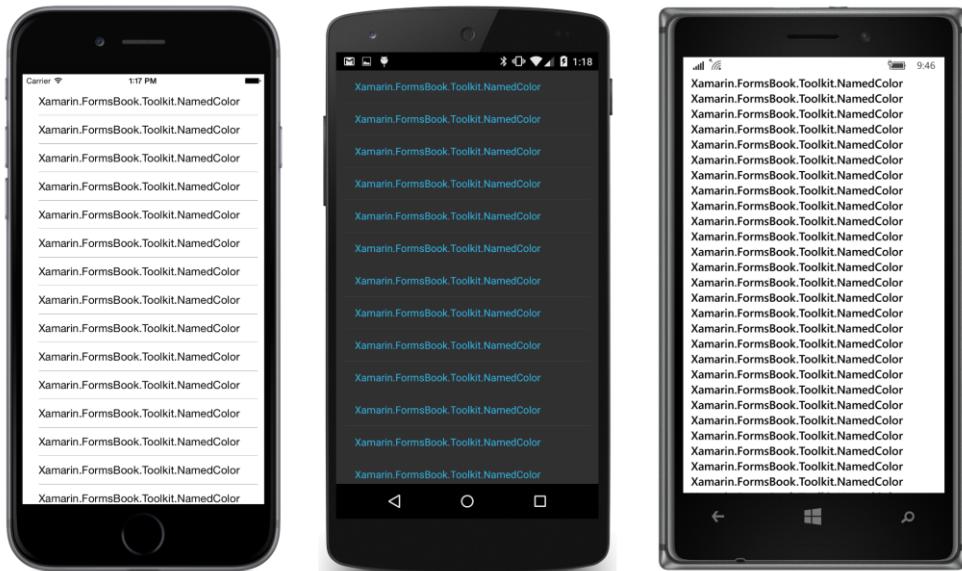
<ListView ItemsSource = "x:Static toolkit:NamedColor.All" />

</ContentPage>

```

Because this program accesses the NamedColor class solely from the XAML file, the program calls `Toolkit.Init` from its App constructor.

You'll discover that you can scroll this list and select items, but the items themselves might be a little disappointing, for what you'll see is a list of 147 fully qualified class names:



This might seem disappointing, but in your future real-life programming work involving ListView, you'll probably cheer when you see something like this display because it means that you've successfully set `ItemsSource` to a valid collection. The objects are there. You just need to display them a little better.

This particular ListView displays the fully qualified class name of NamedColor because NamedColor does not define its own `ToString` method, and the default implementation of `ToString` displays the class name. One simple solution is to add a `ToString` method to NamedColor:

```
public override string ToString()
{
    return FriendlyName;
}
```

Now the ListView displays the friendly names of all the colors. Simple enough.

However, in real-life programming, you might not have the option to add code to your data classes because you might not have access to the source code. So let's pursue solutions that are independent of the actual implementation of the data.

ListView derives from ItemsView, and besides defining the ItemsSource property, ItemsView also defines a property named ItemTemplate of type DataTemplate. The DataTemplate object gives you (the programmer) the power to display the items of your ListView in whatever way you want.

When used in connection with ListView, the DataTemplate references a Cell class to render the items. The Cell class derives from Element, from which it picks up support for parent/child relationships. But unlike View, Cell does not derive from VisualElement. A Cell is more like a *description* of a tree of visual elements rather than a visual element itself.

Here's the class hierarchy showing the five classes that derive from Cell:

```
Object
  BindableObject
  Element
  Cell
    TextCell — two Label views
      ImageCell — derives from TextCell and adds an Image view
      EntryCell — an Entry view with a Label
      SwitchCell — a Switch with a Label
      ViewCell — any View ( likely with children)
```

The descriptions of Cell types are conceptual only: For performance reasons, the actual composition of a Cell is defined within each platform.

As you begin exploring these Cell classes and contemplating their use in connection with ListView, you might question the relevance of a couple of them. But they're not all intended solely for ListView. As you'll see later in this chapter, the Cell classes also play a major role in the TableView, where they are used in somewhat different ways.

The Cell derivatives that have the most applicability to ListView are probably TextCell, ImageCell, and the powerful ViewCell, which lets you define your own visuals for the items.

Let's look at TextCell first, which defines six properties backed by bindable properties:

- Text of type string

- TextColor of type Color
- Detail of type string
- DetailColor of type Color
- Command of type ICommand
- CommandParameter of type Object

The TextCell incorporates two Label views that you can set to two different strings and colors. The font characteristics are fixed in a platform-dependent way.

The **TextCellListCode** program contains no XAML. Instead, it demonstrates how to use a TextCell in code to display properties of all the NamedColor objects:

```
public class TextCellListCodePage : ContentPage
{
    public TextCellListCodePage()
    {
        // Define the DataTemplate.
        DataTemplate dataTemplate = new DataTemplate (typeof (TextCell));
        dataTemplate.SetBinding (TextCell.TextProperty, "FriendlyName");
        dataTemplate.SetBinding (TextCell.DetailProperty,
            new Binding (path: "RgbDisplay", stringFormat: "RGB = {0}"));

        // Build the page.
        Padding = new Thickness (10, Device.OnPlatform(20, 0, 0), 10, 0);

        Content = new ListView
        {
            ItemsSource = NamedColor.All,
            ItemTemplate = dataTemplate
        };
    }
}
```

The first step in using a Cell in a ListView is to create an object of type DataTemplate:

```
DataTemplate dataTemplate = new DataTemplate (typeof (TextCell));
```

Notice that the argument to the constructor is not an *instance* of TextCell but the *type* of TextCell.

The second step is to call a SetBinding method on the DataTemplate object, but notice how these SetBinding calls actually target bindable properties of the TextCell:

```
dataTemplate.SetBinding (TextCell.TextProperty, "FriendlyName");
dataTemplate.SetBinding (TextCell.DetailProperty,
    new Binding (path: "RgbDisplay", stringFormat: "RGB = {0}));
```

These SetBinding calls are identical to bindings that you might set on a TextCell object, but at the time of these calls, there are no instances of TextCell on which to set the bindings!

If you'd like, you can also set some properties of the `TextCell` to constant values by calling the `SetValue` method of the `DataTemplate` class:

```
dataTemplate.SetValue( TextCell .TextColorProperty, Color .Blue);  
dataTemplate.SetValue( TextCell .DetailColorProperty, Color .Red);
```

These `SetValue` calls are similar to calls you might make on visual elements instead of setting properties directly.

The `SetBinding` and `SetValue` methods should be very familiar to you because they are defined by `BindableObject` and inherited by very many classes in Xamarin.Forms. However, `DataTemplate` does not derive from `BindableObject` and instead defines its own `SetBinding` and `SetValue` methods. The purpose of these methods is *not* to bind or set properties of the `DataTemplate` instance. Because `DataTemplate` doesn't derive from `BindableObject`, it has no bindable properties of its own. Instead, `DataTemplate` simply saves these settings in two internal dictionaries that are publicly accessible through two properties that `DataTemplate` defines, named `Bindings` and `Values`.

The third step in using a Cell with `ListView` is to set the `DataTemplate` object to the `ItemTemplate` property of the `ListView`:

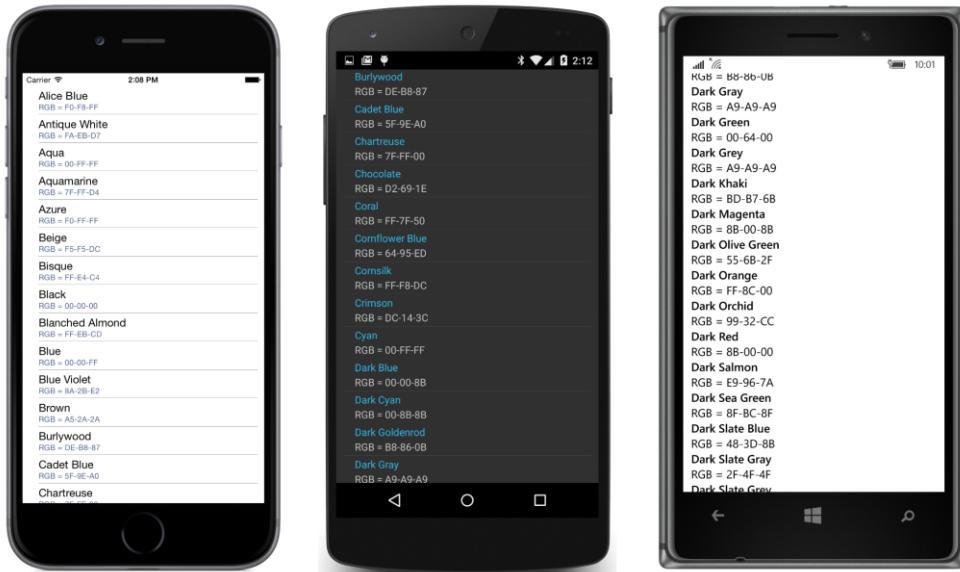
```
Content = new ListView  
{  
    ItemsSource = NamedColor .All,  
    ItemTemplate = dataTemplate  
};
```

Here's what happens (conceptually anyway):

When the `ListView` needs to display a particular item (in this case, a `NamedColor` object), it instantiates the type passed to the `DataTemplate` constructor, in this case a `TextCell`. Any bindings or values that have been set on the `DataTemplate` are then transferred to this `TextCell`.

The Binding-

Context of each `TextCell` is set to the particular item being displayed, which in this case is a particular `NamedColor` object, and that's how each item in the `ListView` displays properties of a particular `NamedColor` object. Each `TextCell` is a visual tree with identical data bindings, but with a unique `BindingContext` setting. Here's the result:



In general, the `ListView` will not create all the visual trees at once. For performance purposes, it will create them only as necessary as the user scrolls new items into view. You can get some sense of this if you install handlers for the `ItemAppearing` and `ItemDisappearing` events defined by

`ListView`. You'll discover that these events don't exactly track the visuals—items are reported as appearing before they scroll into view, and are reported as disappearing after they scroll out of view—but the exercise is instructive nevertheless.

You can also get a sense of what's going on with an alternative constructor for `DataTemplate` that takes a `Func` object:

```
    DataTemplate dataTemplate = new DataTemplate () =>
    {
        return new TextCell ();
    });
}
```

The `Func` object is called only as the `TextCell` objects are required for the items, although these calls actually are made somewhat in advance of the items scrolling into view.

You might want to include code that actually counts the number of `TextCell` instances being created and displays the result in the **Output** window of Visual Studio or Xamarin Studio:

```
int count = 0;
DataTemplate dataTemplate = new DataTemplate () =>
{
    System.Diagnostics.Debug.WriteLine("Text Cell Number " + (++count));
    return new TextCell ();
});
```

As you scroll down to the bottom, you'll discover that a maximum of 147 `TextCell` objects are created for the 147 items in the `ListView`. The `TextCell` objects are cached, but not reused as items scroll in and out of view. However, on a lower level—in particular, involving the platform-specific `TextCellRenderer` objects and the underlying platform-specific visuals created by these renderers—the visuals are reused.

This alternative `DataTemplate` constructor with the `Func` argument might be handy if you need to set some properties on the cell object that you can't set using data bindings. Perhaps you've created a `ViewCell` derivative that requires an argument in its constructor. In general, however, use the constructor with the `Type` argument or define the data template in XAML.

In XAML, the binding syntax somewhat distorts the actual mechanics used to generate visual trees for the `ListView` items, but at the same time the syntax is conceptually clearer and visually more elegant. Here's the XAML file from the `TextCellListXaml` program that is functionally identical to the

`TextCellListCode` program:

```
<ContentPage xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns:x = " http://schemas.microsoft.com/winfx/2009/xaml "
    xmlns:toolkit =
        " clr-namespace:Xamarin.FormsBook.Toolkit;assembly=Xamarin.FormsBook.Toolkit "
    x:Class = "TextCellListXaml.TextCellListXamlPage " >

    <ContentPage.Padding >
        <OnPlatform x:TypeArguments = "Thickness" >
            iOS = "10, 20, 10, 0"
            Android = "10, 0"
            WinPhone = "10, 0" />
    </ContentPage.Padding >

    <ListView ItemsSource = " {x:Static toolkit:NamedColor.All} " >
        <ListView.ItemTemplate >
            <DataTemplate >
                <TextCell Text = " {Binding FriendlyName} "
                    Detail = " {Binding RgbDisplay, StringFormat=RGB = {0}} " />
            </DataTemplate >
        </ListView.ItemTemplate >
    </ListView >
</ContentPage >
```

In XAML, set a `DataTemplate` to the `ItemTemplate` property of the `ListView` and define `TextCell` as a child of `DataTemplate`. Then simply set the data bindings on the `TextCell` properties as if the `TextCell` were a normal visual element. These bindings don't need `Source` settings because a `BindingContext` has been set on each item by the `ListView`.

You'll appreciate this syntax even more when you define your own custom cells.

Custom cells

One of the classes that derives from `Cell` is named `ViewCell`, which defines a single property named `View` that lets you define a custom visual tree for the display of items in a `ListView`.

There are several ways to define a custom cell, but some are less pleasant than others. Perhaps the greatest amount of work involves mimicking the existing Cell classes, which doesn't involve ViewCell at all but instead requires that you create platform-specific cell renderers. You can alternatively derive a class from ViewCell, define several bindable properties of that class similar to the bindable properties of TextCell and the other Cell derivatives, and define a visual tree for the cell in either XAML or code, much as you would do for a custom view derived from ContentView. You can then use that custom cell in code or XAML just like TextCell.

If you want to do the job entirely in code, you can use the DataTemplate constructor with the Func argument and build the visual tree in code as each item is requested. This approach allows you to define the data bindings as the visual tree is being built instead of setting bindings on the DataTemplate.

But certainly the easiest approach is defining the visual tree and bindings of the cell right in XAML within the ListView element. The **CustomNamedColorList** program demonstrates this technique. Everything is in the XAML file:

```
<ContentPage xmlns = "http://xamarin.com/schemas/2014/forms"
              xmlns:x = "http://schemas.microsoft.com/winfx/2009/xaml"
              xmlns:toolkit =
                "clr-namespace:Xamarin.FormsBook.Toolkit;assembly=Xamarin.FormsBook.Toolkit"
              x:Class = "CustomNamedColorList.CustomNamedColorListPage">

  <ContentPage.Padding>
    <OnPlatform x:TypeArguments = "Thickness">
      iOS = "10, 20, 10, 0"
      Android = "10, 0"
      WinPhone = "10, 0" />
  </ContentPage.Padding>

  <ListView SeparatorVisibility = "None">
    ItemsSource = "{x:Static toolkit:NamedColor.All}"
    <ListView.RowHeight>
      <OnPlatform x:TypeArguments = "x:Int32">
        iOS = "80"
        Android = "80"
        WinPhone = "80" />
    </ListView.RowHeight>

    <ListView.ItemTemplate>
      <DataTemplate>
        <ViewCell>
          <ContentView Padding = "5">
            <Frame OutlineColor = "Accent"
                  Padding = "10">
              <StackLayout Orientation = "Horizontal">
                <BoxView x:Name = "boxView"
                        Color = "{Binding Color}"
                        WidthRequest = "50"
                        HeightRequest = "50"/>
              <StackLayout>
                <Label Text = "{Binding FriendlyName}">
              </StackLayout>
            </Frame>
          </ContentView>
        </ViewCell>
      </DataTemplate>
    </ListView.ItemTemplate>
  </ListView>
</ContentPage>
```

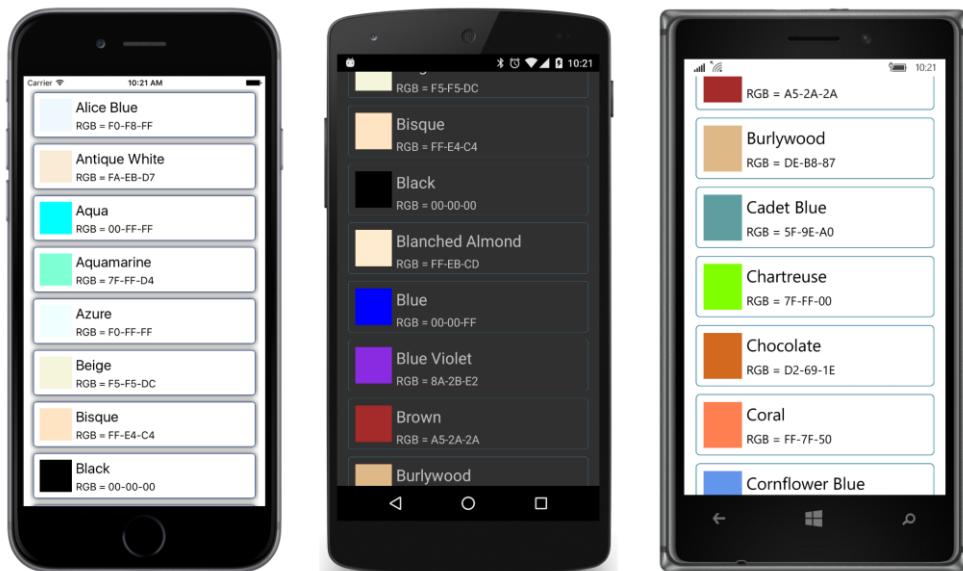
```

        FontSize = "22"
        VerticalOptions = "StartAndExpand" />
    <Label Text = "{Binding RgbDisplay, StringFormat=RGB = {0}}"
        FontSize = "16"
        VerticalOptions = "CenterAndExpand" />
    </StackLayout>
    </StackLayout>
    </Frame>
</ContentView>
</ViewCell>
</DataTemplate>
</ListView.ItemTemplate>
</ListView>
</ContentPage>

```

Within the DataTemplate property-element tags is a ViewCell. The content property of ViewCell is View, so you don't need ViewCell.View tags. Instead, a visual tree within the ViewCell tags is implicitly set to the View property. The visual tree begins with a ContentView to add a little padding, then a Frame and a pair of nested StackLayout elements with a BoxView and two Label elements. When the ListView renders its items, the BindingContext for each displayed item is the item itself, so the Binding markup extensions are generally very simple.

Notice that the RowHeight property of the ListView is set with property element tags for platform-dependent values. These values here were obtained empirically by trial and error, and result in the following displays:



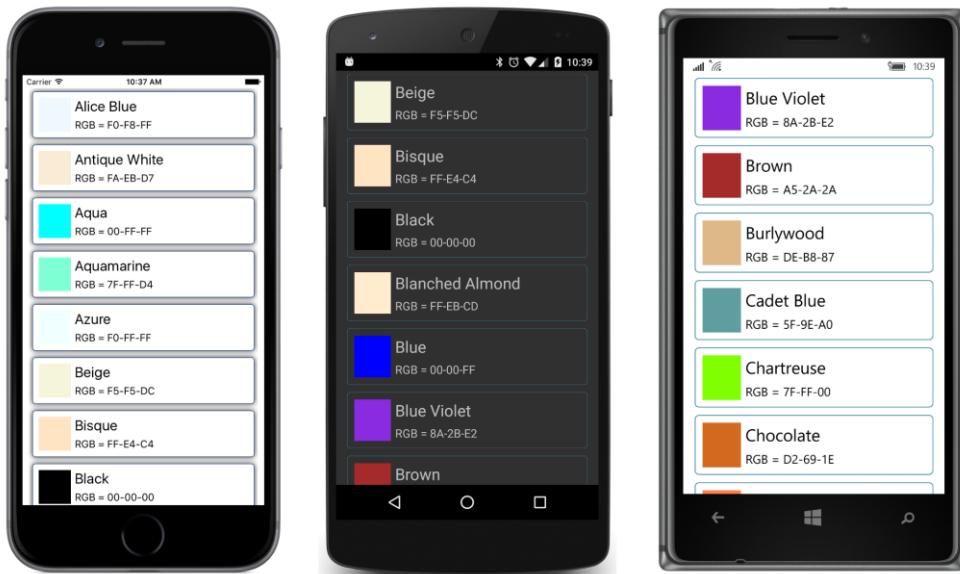
Throughout this book, you have seen several scrollable lists of colors, such as the **ColorBlocks** program in Chapter 4, “Scrolling the stack,” and the **ColorViewList** program in Chapter 8, “Code and XAML in harmony,” but I think you’ll agree that this is the most elegant solution to the problem.

Explicitly setting the RowHeight property of the ListView is one of two ways to set the height of the rows. You can experiment with another approach by removing the RowHeight setting and instead setting the HasUnevenRows property to True. Here's a variation of the CustomNamedColorList program:

```
<ListView SeparatorVisibility = "None" 
    ItemsSource = "(x:Static toolkit:NamedColor.All)" 
    HasUnevenRows = "True" >

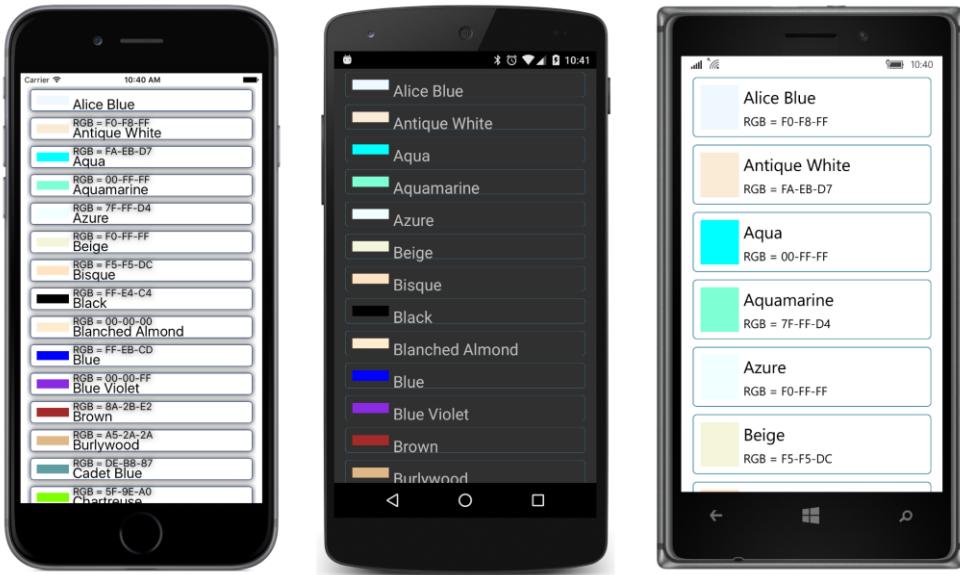
    <ListView.ItemTemplate>
        ...
    </ListView.ItemTemplate>
</ListView>
```

The HasUnevenRows property is designed specifically to handle cases when the heights of the cells in the ListView are not uniform. However, you can also use it for cases when all the cells are the same height but you don't know precisely what that height is. With this setting, the heights of the individual rows are calculated based on the visual tree, and that height is used to space the rows. In this example, the heights of the cells are governed by the heights of the two Label elements. The rows are just a little different than the heights explicitly set from the RowHeight property:



Although the HasUnevenRows property seems to provide an easier approach to sizing cell heights than RowHeight, it does have a performance penalty and you should avoid it unless you need it.

But for iOS and Android, you must use one or the other of the two properties when defining a custom cell. Here's what happens when neither property is set:



Only the Windows platforms automatically use the rendered size of the visual tree to determine the row height.

In summary, for best ListView performance, use one of the predefined Cell classes. If you can't, use ViewCell and define your own visual tree. Try your best to supply a specific RowHeight property setting with ViewCell. Use HasUnevenRows only when that is not possible.

Grouping the ListView items

It's sometimes convenient for the items in a ListView to be grouped in some way. For example, a ListView that lists the names of a user's friends or contacts is easily navigable if the items are in alphabetical order, but it's even more navigable if all the A's, B's, C's, and so forth are in separate groups, and a few taps are all that's necessary to navigate to a particular group.

The ListView supports such grouping and navigation.

As you've discovered, the object you set to the ItemsSource property of ListView must implement `IEnumerable`. This `IEnumerable` object is a collection of items.

When using ListView with the grouping feature, the `IEnumerable` collection you set to Items-Source contains one item for each group, and these items themselves implement `IEnumerable` and contain the objects in that group. In other words, you set the ItemsSource property of ListView to a collection of collections.

One easy way for the group class to implement `IEnumerable` is to derive from `List` or `ObservableCollection`, depending on whether items can be dynamically added to or removed from the collection. However, you'll want to add a couple of other properties to this class: One property (typically

called **Title**) should be a text description of the group. Another property is a shorter text description that's used to navigate the list. Based on how this text description is used on Windows 10 Mobile, you should keep this short text description to three letters or fewer.

For example, suppose you want to display a list of colors but divided into groups indicating the dominant hue (or lack of hue). Here are seven such groups: grays, reds, yellows, greens, cyans, blues, and magentas.

The **NamedColorGroup** class in the **Xamarin.FormsBook.Toolkit** library derives from **List<NamedColor>** and hence is a collection of **NamedColor** objects. It also defines text **Title** and **ShortName** properties and a **ColorShade** property intended to serve as a pastel-like representative color of the group:

```
public class NamedColorGroup : List < NamedColor >
{
    // Instance members.

    private NamedColorGroup( string title, string shortName, Color colorShade )
    {
        this .Title = title;
        this .ShortName = shortName;
        this .ColorShade = colorShade;
    }

    public string Title { private set ; get ; }

    public string ShortName { private set ; get ; }

    public Color ColorShade { private set ; get ; }

    // Static members.

    static NamedColorGroup()
    {
        // Create all the groups.
        List < NamedColorGroup > groups = new List < NamedColorGroup >
        {
            new NamedColorGroup ( "Grays" , "Gry" , new Color ( 0.75 , 0.75 , 0.75 ) ),
            new NamedColorGroup ( "Reds" , "Red" , new Color ( 1 , 0.75 , 0.75 ) ),
            new NamedColorGroup ( "Yellows" , "Yel" , new Color ( 1 , 1 , 0.75 ) ),
            new NamedColorGroup ( "Greens" , "Gm" , new Color ( 0.75 , 1 , 0.75 ) ),
            new NamedColorGroup ( "Cyan" , "Cyn" , new Color ( 0.75 , 1 , 1 ) ),
            new NamedColorGroup ( "Blues" , "Blu" , new Color ( 0.75 , 0.75 , 1 ) ),
            new NamedColorGroup ( "Magentas" , "Mag" , new Color ( 1 , 0.75 , 1 ) )
        };
    }

    foreach ( NamedColor namedColor in NamedColor .All )
    {
        Color color = namedColor.Color;
        int index = 0;

        if ( color.Saturation != 0 )
        {
            index = 1 + ( int )( ( 12 * color.Hue + 1 ) / 2 ) % 6;
        }
    }
}
```

```
        }

        groups[index].Add(namedColor);

    }

}

foreach ( NamedColorGroup group in groups)
{
    group.TrimExcess();
}

All = groups;

}

public static IList< NamedColorGroup > All { private set ; get ; }
```

A static constructor assembles seven `NamedColorGroup` instances and sets the static `All` property to the collection of these seven objects.

The **ColorGroupList** program uses this new class for its **ListView**. Notice that the **ItemsSource** is set to **NamedColorGroup.All** (a collection of seven items) rather than **NamedColor.All** (a collection of 147 items).

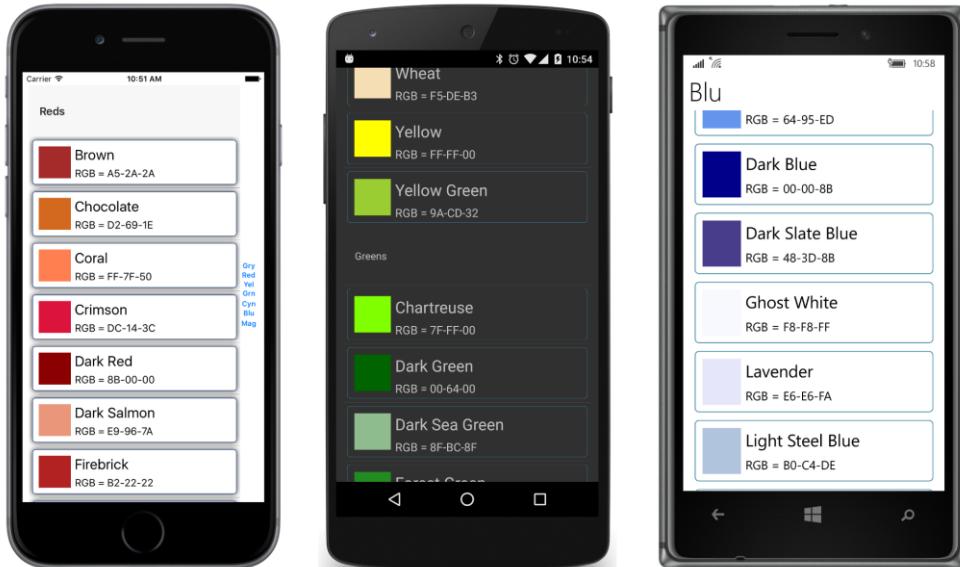
```

        Color = "{Binding Color}"
        WidthRequest = "50"
        HeightRequest = "50" />
    </StackLayout>
    <Label Text = "{Binding FriendlyName}" 
        FontSize = "22"
        VerticalOptions = "StartAndExpand" />
    <Label Text = "{Binding RgbDisplay, StringFormat=RGB={0}}" 
        FontSize = "16"
        VerticalOptions = "CenterAndExpand" />
</StackLayout>
</Frame>
</ContentView>
</ViewCell>
</DataTemplate>
</ListView.ItemTemplate>
</ListView>
</ContentPage>

```

Setting IsGroupingEnabled to True is very important. Remove that (as well as the ItemTemplate setting), and the ListView displays seven items identified by the fully qualified class name "Xamarin.FormsBook.Toolkit.NamedColorGroup".

The GroupDisplayBinding property is a Binding referencing the name of a property in the group items that contains a heading or title for the group. This is displayed in the ListView to identify each group:



The GroupShortNameBinding property is bound to another property in the group objects that displays a condensed version of the header. If the group headings are just the letters A, B, C, and so

forth, you can use the same property for the short names.

On the iPhone screen, you can see the short names at the right side of the screen. In iOS terminology, this is called an *index* for the list, and tapping one moves to that part of the list.

On the Windows 10 Mobile screen, the headings incorrectly use the `ShortName` rather than the `Title` property. Tapping a heading goes to a navigation screen (called a *jump list*) where all the short names are arranged in a grid. Tapping one goes back to the `ListView` with the corresponding header at the top of the screen.

Android provides no navigation.

Even though the `ListView` is now really a collection of `NamedColorGroup` objects, `SelectedItem` is still a `NamedColor` object.

In general, if an `ItemSelected` handler needs to determine the group of a selected item, you can do that “manually” by accessing the collection set to the `ItemsSource` property and using one of the `Find methods defined by List`. Or you can store a group identifier within each item. The `Tapped` handler provides the group as well as the item.

Custom group headers

If you don't like the particular style of the group headers that Xamarin.Forms supplies, there's something you can do about it.

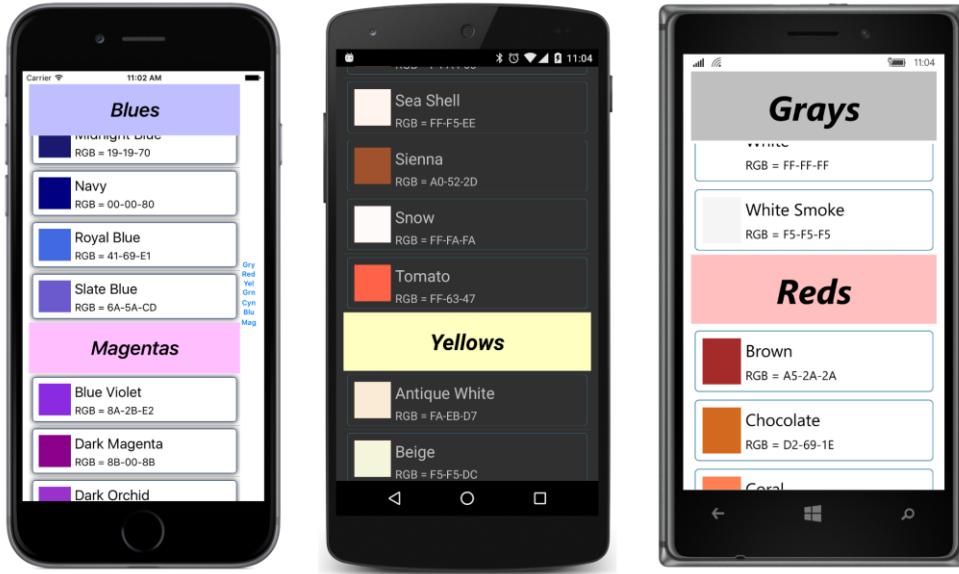
Rather than setting a binding to the `GroupDisplayBinding` property, set a

`DataTemplate` to the `GroupHeaderTemplate` property:

```
< ListView ItemsSource = " {x:Static toolkit:NamedColorGroup.All} "
    IsGroupingEnabled = " True "
    GroupShortNameBinding = " {Binding ShortName} " >

    ...
    < ListView.GroupHeaderTemplate >
        < DataTemplate >
            < ViewCell >
                < Label Text = " {Binding Title} "
                    BackgroundColor = " {Binding ColorShade} "
                    TextColor = " Black "
                    FontAttributes = " Bold,italic "
                    HorizontalTextAlignment = " Center "
                    VerticalTextAlignment = " Center " >
                    < Label.FontSize >
                        < OnPlatform x:TypeArguments = " x:Double "
                            iOS = " 30 "
                            Android = " 30 "
                            WinPhone = " 45 " />
                    </ Label.FontSize >
                </ Label >
            </ ViewCell >
        </ DataTemplate >
    </ ListView.GroupHeaderTemplate >
</ ListView >
```

Notice that the Label has a fixed text color of black, so the `BackgroundColor` property should be set to something light that provides a good contrast with the text. Such a color is available from the `NamedColorGroup` class as the `ColorShade` property. This allows the background of the header to reflect the dominant hue associated with the group:



Notice how the header for the topmost item remains fixed at the top on iOS and Windows 10 Mobile and scrolls off the top of the screen only when another header replaces it.

ListView and interactivity

An application can interact with its `ListView` in a variety of ways: If the user taps an item, the `ListView` fires an `ItemTapped` event and, if the item is previously not selected, also an `ItemSelected` event. A program can also define a data binding by using the `SelectedItem` property. The `ListView` has a `ScrollTo` method that lets a program scroll the `ListView` to make a particular item visible. Later in this chapter you'll see a refresh facility implemented by `ListView`.

Cell itself defines a `Tapped` event, but you'll probably use that event in connection with `Table-View` rather than `ListView`. `TextCell` defines the same `Command` and `CommandParameter` properties as `Button` and `ToolbarItem`, but you'll probably use those properties in connection with `Table-View` as well. You can also define a context menu on a cell; this is demonstrated in the section "Context menus" later in this chapter.

It is also possible for a Cell derivative to contain some interactive views. The `EntryCell` and `SwitchCell` allow the user to interact with an `Entry` or a `Switch`. You can also include interactive views in a `ViewCell`.

The **InteractiveListView** program contains in its XAML file a ListView named listView. The code-behind file sets the ItemsSource property of that ListView to a collection of type List<ColorViewModel>, containing 100 instances of ColorViewModel—a class described in Chapter 18, “MVVM,” and which can be found in the **Xamarin.FormsBook.Toolkit** library. Each instance of ColorViewModel is initialized to a random color:

```
public partial class InteractiveListViewPage : ContentPage
{
    public InteractiveListViewPage()
    {
        InitializeComponent();

        const int count = 100;
        List<ColorViewModel> colorList = new List<ColorViewModel>(count);
        Random random = new Random();

        for (int i = 0; i < count; i++)
        {
            ColorViewModel colorViewModel = new ColorViewModel();
            colorViewModel.Color = new Color(random.NextDouble(),
                random.NextDouble(),
                random.NextDouble());
            colorList.Add(colorViewModel);
        }
        listView.ItemsSource = colorList;
    }
}
```

The ListView in the XAML file contains a data template using a ViewCell that contains three Slider views, a BoxView, and a few Label elements to display the hue, saturation, and luminosity values, all of which are bound to properties of the ColorViewModel class:

```
<ContentPage xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns:x = " http://schemas.microsoft.com/winfx/2009/xaml "
    xmlns:toolkit =
        " clr-namespace:Xamarin.FormsBook.Toolkit;assembly=Xamarin.FormsBook.Toolkit "
    x:Class = " InteractiveListView.InteractiveListViewPage " >

    <ContentPage.Padding >
        <OnPlatform x:TypeArguments = " Thickness " >
            iOS = " 10, 20, 10, 0 "
            Android = " 10, 0 "
            WinPhone = " 10, 0 " />
    </ContentPage.Padding >

    <ContentPage.Resources >
        <ResourceDictionary >
            <toolkit:ColorToContrastColorConverter x:Key = " contrastColor " />
        </ResourceDictionary >
    </ContentPage.Resources >

    <ListView x:Name = " listView "
        HasUnevenRows = " True " >
        <ListView.ItemTemplate >
```

```
< DataTemplate >
  < ViewCell >
    < Grid Padding = "0, 5" >
      < Grid.RowDefinitions >
        < RowDefinition Height = "Auto" />
        < RowDefinition Height = "Auto" />
        < RowDefinition Height = "Auto" />
      </ Grid.RowDefinitions >

      < Grid.ColumnDefinitions >
        < ColumnDefinition Width = "*" />
        < ColumnDefinition Width = "Auto" />
      </ Grid.ColumnDefinitions >

      < Slider Value = "{Binding Hue, Mode=TwoWay}"
               Grid.Row = "0" Grid.Column = "0" />

      < Slider Value = "{Binding Saturation, Mode=TwoWay}"
               Grid.Row = "1" Grid.Column = "0" />

      < Slider Value = "{Binding Luminosity, Mode=TwoWay}"
               Grid.Row = "2" Grid.Column = "0" />

      < ContentView BackgroundColor = "{Binding Color}"
                    Grid.Row = "0" Grid.Column = "1" Grid.RowSpan = "3"
                    Padding = "10" >

        < StackLayout Orientation = "Horizontal"
                      VerticalOptions = "Center" >
          < Label Text = "{Binding Hue, StringFormat={0:F2}, }"
                  TextColor = "{Binding Color,
                  Converter={StaticResource contrastColor}}" />

          < Label Text = "{Binding Saturation, StringFormat={0:F2}, }"
                  TextColor = "{Binding Color,
                  Converter={StaticResource contrastColor}}" />

          < Label Text = "{Binding Luminosity, StringFormat={0:F2}, }"
                  TextColor = "{Binding Color,
                  Converter={StaticResource contrastColor}}" />
        </ StackLayout >
      </ ContentView >
    </ Grid >
  </ ViewCell >
</ DataTemplate >
</ ListView.ItemTemplate >
</ ListView >
</ ContentPage >
```

The Label elements sit on top of the BoxView, so they should be made a color that contrasts with the background. This is accomplished with the `ColorToContrastColorConverter` class (also in `Xamarin.FormsBook.Toolkit`), which calculates the luminance of the color by using a standard formula and then converts to `Color.Black` for a light color and `Color.White` for a dark color:

```
namespace Xamarin.FormsBook.Toolkit
{
    public class ColorToContrastColorConverter : IValueConverter
    {
        public object Convert( object value, Type targetType,
            object parameter, CultureInfo culture)
        {
            return ColorToContrastColor(( Color )value);
        }

        public object ConvertBack( object value, Type targetType,
            object parameter, CultureInfo culture)
        {
            return ColorToContrastColor(( Color )value);
        }

        Color ColorToContrastColor( Color color)
        {
            // Standard luminance calculation.
            double luminance = 0.30 * color.R +
                0.59 * color.G +
                0.11 * color.B;

            return luminance > 0.5 ? Color .Black : Color .White;
        }
    }
}
```

Here's the result:



Each of the items independently lets you manipulate the three Slider elements to select a new

color, and while this example might seem a little artificial, a real-life example involving a collection of identical visual trees is not inconceivable. Even if there are just a few items in the collection, it might make sense to use a `ListView` that displays all the items on the screen and doesn't scroll. `ListView` is one of the most powerful tools that XAML provides to compensate for its lack of programming loops.

ListView and MVVM

`ListView` is one of the major players in the View part of the Model-View-ViewModel architecture. Whenever a `ViewModel` contains a collection, a `ListView` generally displays the items.

A collection of ViewModels

Let's explore the use of `ListView` in MVVM with some data that more closely approximates a real-life example. This is a collection of information about 65 fictitious students of the fictitious School of Fine Art, including images of their overly spherical heads.

These images and an XML file containing the student names and references to the bitmaps are in a website at <http://xamarin.github.io/xamarin-forms/samples/students/>. This website is hosted from the same GitHub repository as the source code for this book, and the contents of the site can be found in the `gh-pages` branch of that repository.

The `Students.xml` file at that site contains information about the school and students. Here's the beginning and the end with abbreviated URLs of the photos.

```
< StudentBody xmlns:xsi =http://www.w3.org/2001/XMLSchema-instance
  xmlns:xsd = "http://www.w3.org/2001/XMLSchema" >
  < School > School of Fine Art </ School >
  < Students >
    < Student >
      < FullName > Adam Harmetz </ FullName >
      < FirstName > Adam </ FirstName >
      < MiddleName />
      < LastName > Harmetz </ LastName >
      < Sex > Male </ Sex >
      < PhotoFilename > http://xamarin.github.io/.../AdamHarmetz.png </ PhotoFilename >
      < GradePointAverage > 3.01 </ GradePointAverage >
    </ Student >
    < Student >
      < FullName > Alan Brewer </ FullName >
      < FirstName > Alan </ FirstName >
      < MiddleName />
      < LastName > Brewer </ LastName >
      < Sex > Male </ Sex >
      < PhotoFilename > http://xamarin.github.io/.../AlanBrewer.png </ PhotoFilename >
      < GradePointAverage > 1.17 </ GradePointAverage >
    </ Student >
    ...
  < Student >
```

```

< FullName > Tzipi Butnaru </ FullName >
< FirstName > Tzipi </ FirstName >
< MiddleName >
< LastName > Butnaru </ LastName >
< Sex > Female </ Sex >
< PhotoFilename > http://xamarin.github.io/.../TzipiButnaru.png </ PhotoFilename >
< GradePointAverage > 3.76 </ GradePointAverage >
</ Student >
< Student >
< FullName > Zrinka Makovac </ FullName >
< FirstName > Zrinka </ FirstName >
< MiddleName >
< LastName > Makovac </ LastName >
< Sex > Female </ Sex >
< PhotoFilename > http://xamarin.github.io/.../ZrinkaMakovac.png </ PhotoFilename >
< GradePointAverage > 2.73 </ GradePointAverage >
</ Student >
</ Students >
</ StudentBody >

```

The grade point averages were randomly generated when this file was created.

In the **Libraries** directory among the source code for this book, you'll find a library project named **SchoolOfFineArt** that accesses this XML file and uses XML deserialization to convert it into classes named **Student**, **StudentBody**, and **SchoolViewModel**. Although the **Student** and **StudentBody** classes don't have the words **ViewModel** in their names, they qualify as **ViewModels** regardless.

The **Student** class derives from **ViewModelBase** (a copy of which is included in the **SchoolOfFineArt** library) and defines the seven properties associated with each **Student** element in the XML file. An eighth property is used in a future chapter. The class also defines four additional properties of type **ICommand** and a final property named **StudentBody**. These final five properties are not set from the XML deserialization, as the **Xmllgnore** attributes indicate:

```

namespace SchoolOfFineArt
{
    public class Student : ViewModelBase
    {
        string fullName, firstName, middleName;
        string lastName, sex, photoFilename;
        double gradePointAverage;
        string notes;

        public Student()
        {
            ResetGpaCommand = new Command(() => GradePointAverage = 2.5m);
            MoveToTopCommand = new Command(() => StudentBody.MoveStudentToTop(this));
            MoveToBottomCommand = new Command(() => StudentBody.MoveStudentToBottom(this));
            RemoveCommand = new Command(() => StudentBody.RemoveStudent(this));
        }

        public string FullName
        {

```

```
        set { SetProperty( ref fullName, value ); }
        get { return fullName; }
    }

    public string FirstName
    {
        set { SetProperty( ref firstName, value ); }
        get { return firstName; }
    }

    public string MiddleName
    {
        set { SetProperty( ref middleName, value ); }
        get { return middleName; }
    }

    public string LastName
    {
        set { SetProperty( ref lastName, value ); }
        get { return lastName; }
    }

    public string Sex
    {
        set { SetProperty( ref sex, value ); }
        get { return sex; }
    }

    public string PhotoFilename
    {
        set { SetProperty( ref photoFilename, value ); }
        get { return photoFilename; }
    }

    public double GradePointAverage
    {
        set { SetProperty( ref gradePointAverage, value ); }
        get { return gradePointAverage; }
    }

    // For program in Chapter 25.

    public string Notes
    {
        set { SetProperty( ref notes, value ); }
        get { return notes; }
    }

    // Properties for implementing commands.

    [XmlIgnore]
    public ICommand ResetGpaCommand { private set ; get ; }

    [XmlIgnore]
    public ICommand MoveToTopCommand { private set ; get ; }
```

```
[ XmlIgnore ]
public ICommand MoveToBottomCommand { private set; get; }

[ XmlIgnore ]
public ICommand RemoveCommand { private set; get; }

[ XmlIgnore ]
public StudentBody StudentBody { set; get; }

}
```

The four properties of type `ICommand` are set in the `Student` constructor and associated with short methods, three of which call methods in the `StudentBody` class. These will be discussed in more detail later.

The `StudentBody` class defines the `School` and `Students` properties. The constructor initializes the `Students` property as an `ObservableCollection` object. In addition, `StudentBody`

defines three methods called from the `Student` class that can remove a student from the list or move a student to the top or bottom of the list:

```
namespace SchoolOfflineArt
{
    public class StudentBody : ViewModelBase
    {
        string school;
        ObservableCollection<Student> students = new ObservableCollection<Student>();

        public string School
        {
            set { SetProperty(ref school, value); }
            get { return school; }
        }

        public ObservableCollection<Student> Students
        {
            set { SetProperty(ref students, value); }
            get { return students; }
        }

        // Methods to implement commands to move and remove students.
        public void MoveStudentToTop( Student student )
        {
            Students.Move(Students.IndexOf(student), 0);
        }

        public void MoveStudentToBottom( Student student )
        {
            Students.Move(Students.IndexOf(student), Students.Count - 1);
        }

        public void RemoveStudent( Student student )
        {
            Students.Remove(student);
        }
    }
}
```

```
        }
    }
}
```

The `SchoolViewModel` class is responsible for loading the XML file and deserializing it. It contains a single property named `StudentBody`, which corresponds to the root tag of the XAML file. This property is set to the `StudentBody` object obtained from the `Deserialize` method of the `XmlSerializer` class.

```
namespace SchoolOfFineArt
{
    public class SchoolViewModel : ViewModelBase
    {
        StudentBody studentBody;
        Random rand = new Random();

        public SchoolViewModel() : this( null )
        {
        }

        public SchoolViewModel( IDictionary< string , object > properties )
        {
            // Avoid problems with a null or empty collection.
            StudentBody = new StudentBody();
            StudentBody.Students.Add( new Student() );

            string uri = "http://xamarin.github.io/xamarin-forms-book-samples" +
                "/SchoolOfFineArt/students.xml";

            HttpWebRequest request = WebRequest.CreateHttp(uri);

            request.BeginGetResponse((arg) =>
            {
                // Deserialize XML file.
                Stream stream = request.EndGetResponse(arg).GetResponseStream();
                StreamReader reader = new StreamReader(stream);
                XmlSerializer xml = new XmlSerializer( typeof( StudentBody ) );
                StudentBody = xml.Deserialize(reader) as StudentBody;

                // Enumerate through all the students
                foreach ( Student student in StudentBody.Students )
                {
                    // Set StudentBody property in each Student object.
                    student.StudentBody = StudentBody;

                    // Load possible Notes from properties dictionary
                    //           (for program in Chapter 25).
                    if ( properties != null && properties.ContainsKey(student.FullName) )
                    {
                        student.Notes = ( string )properties[student.FullName];
                    }
                }
            }, null );
        }
    }
}
```

```

// Adjust GradePointAverage randomly.
Device.StartTimer(TimeSpan.FromSeconds(0.1),
() =>
{
    if (studentBody != null)
    {
        int index = rand.Next(studentBody.Students.Count);
        Student student = studentBody.Students[index];
        double factor = 1 + (rand.NextDouble() - 0.5) / 5;
        student.GradePointAverage = Math.Round(
            Math.Max(0, Math.Min(5, factor * student.GradePointAverage)), 2);
    }
    return true;
});

// Save Notes in properties dictionary for program in Chapter 25.
public void SaveNotes(IDictionary<string, object> properties)
{
    foreach (Student student in StudentBody.Students)
    {
        properties[student.FullName] = student.Notes;
    }
}

public StudentBody StudentBody
{
    protected set { SetProperty(ref studentBody, value); }
    get { return studentBody; }
}
}
}

```

Notice that the data is obtained asynchronously. The properties of the various classes are not set until sometime after the constructor of this class completes. But the implementation of the INotifyPropertyChanged interface should allow a user interface to react to data that is acquired sometime after the program starts up.

The callback to BeginGetResponse runs in the same secondary thread of execution that is used to download the data in the background. This callback sets some properties that cause PropertyChanged events to fire, which result in updates to data bindings and changes to user-interface objects. Doesn't this mean that user-interface objects are being accessed from a second thread of execution? Shouldn't Device.BeginInvokeOnMainThread be used to avoid that?

Actually, it's not necessary. Changes in ViewModel properties that are linked to properties of userinterface objects via data bindings don't need to be marshalled to the user-interface thread.

The SchoolViewModel class is also responsible for randomly modifying the GradePointAverage property of the students, in effect simulating dynamic data. Because Student implements INotify-

PropertyChanged (by virtue of deriving from ViewModelBase), we should be able to see these values change dynamically when displayed by the ListView.

The **SchoolOffFineArt** library also has a static Library.Init method that your program should call if it's referring to the library only from XAML to ensure that the assembly is properly bound to the application.

You might want to play around with the StudentViewModel class to get a feel for the nested properties and how they are expressed in data bindings. You can create a new Xamarin.Forms project (named Tryout, for example), include the **SchoolOffFineArt** project in the solution, and add a reference from Tryout to the **SchoolOffFineArt** library. Then create a ContentPage that looks something like this:

```
<ContentPage xmlns = "http://xamarin.com/schemas/2014/forms"
             xmlns:x = "http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:school = "clr-namespace:SchoolOffFineArt;assembly=SchoolOffFineArt"
             x:Class = "Tryout.TryoutListPage" >

    <ContentPage.Padding>
        <OnPlatform x:TypeArguments = "Thickness" 
                    iOS = "0, 20, 0, 0" />
    </ContentPage.Padding>

    <ContentPage.BindingContext>
        <school:SchoolViewModel />
    </ContentPage.BindingContext>

    <Label />
</ContentPage>
```

The BindingContext of the page is set to the SchoolViewModel instance, and you can experiment with bindings on the Text property of the Label. For example, here's an empty binding:

```
<Label Text = "{Binding StringFormat={0}}" />
```

That displays the fully qualified class name of the inherited BindingContext:

SchoolOfFineArt.SchoolViewModel

The SchoolViewModel class has one property named StudentBody, so set the Path of the Binding to that:

```
<Label Text = "{Binding Path=StudentBody, StringFormat={0}}" />
```

Now you'll see the fully-qualified name of the StudentBody class:

SchoolOfFineArt.StudentBody

The StudentBody class has two properties, named School and Students. Try the School property:

```
<Label Text = "{Binding Path=StudentBody.School,
                           StringFormat={0}}" />
```

Finally, some actual data is displayed rather than just a class name. It's the string from the XML file set to the School property:

School of Fine Art

The StringFormat isn't required in the Binding expression because the property is of type string.

Now try the Students property:

```
<Label Text = "{Binding Path=StudentBody.Students,
StringFormat='{0}'}" />
```

This displays the fully qualified class name of ObservableCollection with a collection of Student objects:

`System.Collections.ObjectModel.ObservableCollection`1[SchoolOfFineArt.Student]`

It should be possible to index this collection, like so:

```
<Label Text = "{Binding Path=StudentBody.Students[0],
StringFormat='{0}'}" />
```

That is an object of type Student:

SchoolOfFineArt.Student

If the entire Students collection is loaded at the time of this binding, you should be able to specify any index on the Students collection, but an index of 0 is always safe.

You can then access a property of that Student, for example:

```
<Label Text = "{Binding Path=StudentBody.Students[0].FullName,
StringFormat='{0}'}" />
```

And you'll see that student's name:

Adam Harmetz

Or, try the GradePointAverage property:

```
<Label Text = "{Binding Path=StudentBody.Students[0].GradePointAverage,
StringFormat='{0}'}" />
```

Initially you'll see the randomly generated value stored in the XML file:

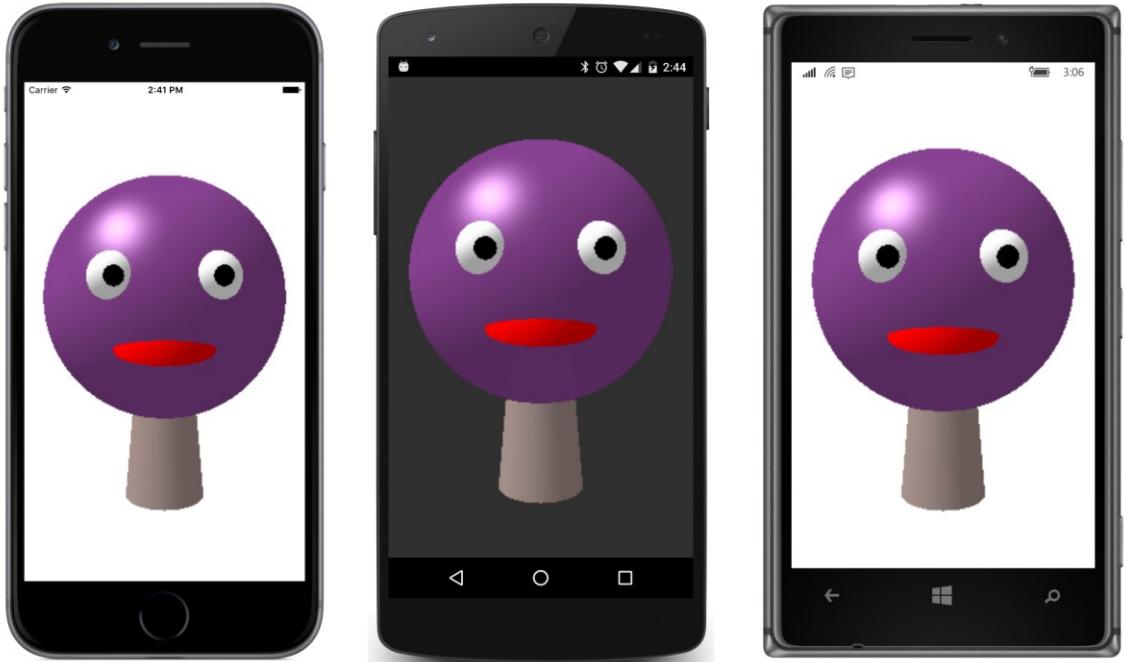
3.01

But wait a little while and you should see it change.

Would you like to see a picture of Adam Harmetz? Just change the Label to an Image, and change the target property to Source and the source path to PhotoFilename:

```
<Image Source = "{Binding Path=StudentBody.Students[0].PhotoFilename}" />
```

And there he is, from the class of 2019:



With that understanding of data-binding paths, it should be possible to construct a page that contains both a Label that displays the name of the school and a ListView that displays all the students with their full names, grade-point averages, and photos. Each item in the ListView must display two pieces of text and an image. This is ideal for an ImageCell, which derives from TextCell and adds an image to the two text items. Here is the **StudentList** program:

```
<ContentPage xmlns = "http://xamarin.com/schemas/2014/forms"
    xmlns:x = "http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:school = "clr-namespace:SchoolOfFineArt;assembly=SchoolOfFineArt"
    x:Class = "StudentList.StudentListPage" >

    <ContentPage.Padding >
        <OnPlatform x:TypeArguments = "Thickness" 
            iOS = "0, 20, 0, 0" />
    </ContentPage.Padding >

    <ContentPage.BindingContext >
        <school:SchoolViewModel />
    </ContentPage.BindingContext >

    <StackLayout BindingContext = "{Binding StudentBody}" >
        <Label Text = "{Binding School}" 
            FontSize = "Large" 
            FontAttributes = "Bold" >
```

```

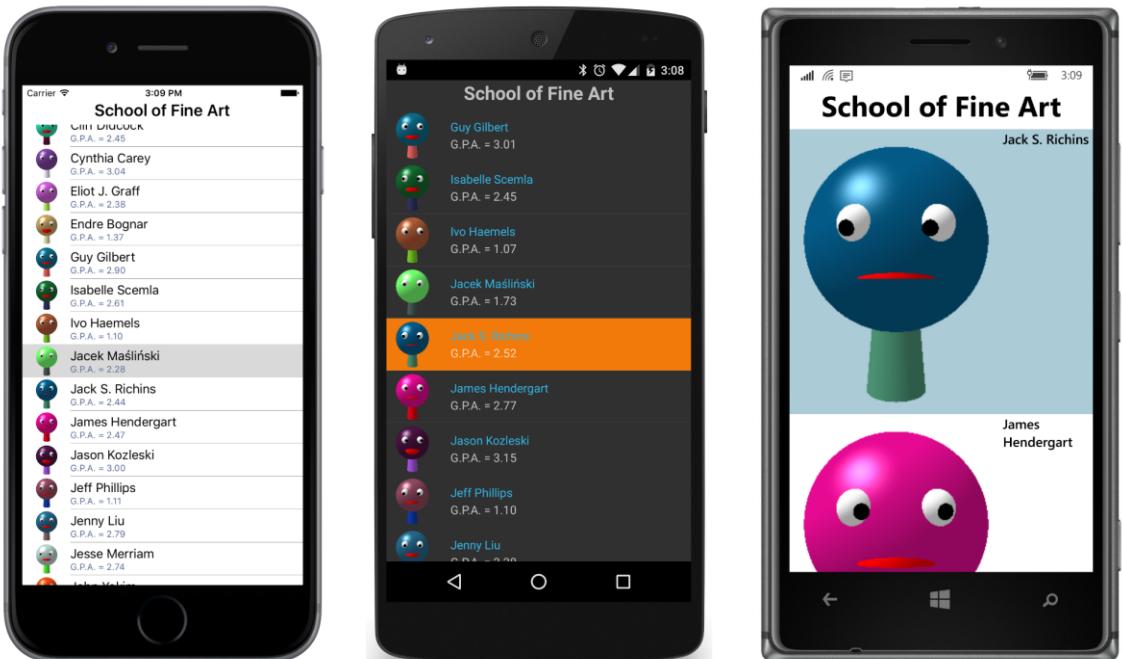
    HorizontalTextAlignment = "Center" />

    < ListView ItemsSource = "{Binding Students}" >
        < ListView.ItemTemplate >
            < DataTemplate >
                < ImageCell ImageSource = "{Binding PhotoFilename}" /
                    Text = "{Binding FullName}" *
                    Detail = "{Binding GradePointAverage,
                        StringFormat='G.P.A. = {0:F2}' }" />
            </ DataTemplate >
        </ ListView.ItemTemplate >
    </ ListView >
</ StackLayout >
</ ContentPage >

```

As in the experimental XAML file, the BindingContext of the ContentPage is the SchoolView-Model object. The StackLayout inherits that BindingContext but sets its own BindingContext to the StudentBody property, and that's the BindingContext inherited by the children of the StackLayout. The Text property of the Label is bound to the School property of the StudentBody class, and the ItemsSource property of the ListView is bound to the Students collection.

This means that the BindingContext for each of the items in the ListView is a Student object, and the ImageCell properties can be bound to properties of the Student class. The result is scrollable and selectable, although the selection is displayed in a platform-specific manner:



Unfortunately, the Windows Runtime version of the ImageCell works a little differently from those

on the other two platforms. If you don't like the default size of these rows, you might be tempted to set the RowHeight property, but it doesn't work in the same way across the platforms, and the only consistent solution is to switch to a custom ViewCell derivative, perhaps one much like the one in

`CustomNamedColorList` but with an `Image` rather than a `BoxView`.

The Label at the top of the page shares the StackLayout with the ListView so that the Label stays in place as the ListView is scrolled. However, you might want such a header to scroll with the contents of the ListView, and you might want to add a footer as well. The ListView has Header and Footer properties of type object that you can set to a string or an object of any type (in which case the header will display the results of that object's `ToString` method) or to a binding.

Here's one approach: The BindingContext of the page is set to the `SchoolViewModel` as before, but the BindingContext of the ListView is set to the `StudentBody` property. This means that the `ItemsSource` property can reference the `Students` collection in a binding, and the `Header` can be bound to the `School` property:

```
<ContentPage ...>
...
<ContentPage.BindingContext>
    <school:SchoolViewModel />
</ContentPage.BindingContext>

<ListView BindingContext = " (Binding StudentBody)"
          ItemsSource = " (Binding Students)"
          Header = " (Binding School)" >
    ...
</ListView>
</ContentPage>
```

That displays the text "School of Fine Art" in a header that scrolls with the ListView content.

If you'd like to format that header, you can do that as well. Set the `HeaderTemplate` property of the ListView to a `DataTemplate`, and within the `DataTemplate` tags define a visual tree. The `BindingContext` for that visual tree is the object set to the `Header` property (in this example, the string with the name of the school).

In the `ListViewHeader` program shown below, the `Header` property is bound to the `School` property. Within the `HeaderTemplate` is a visual tree consisting solely of a `Label`. This `Label` has an empty binding so the `Text` property of that `Label` is bound to the text set to the `Header` property:

```
<ContentPage xmlns = " http://xamarin.com/schemas/2014/forms "
              xmlns:x = " http://schemas.microsoft.com/winfx/2009/xaml "
              xmlns:school = " clr-namespace:SchoolOfFineArt;assembly=SchoolOfFineArt "
              x:Class = " ListViewHeader.ListViewHeaderPage " >

<ContentPage.Padding>
    <OnPlatform x:TypeArguments = " Thickness "
                iOS = " 0, 20, 0, 0 " />
</ContentPage.Padding>

<ContentPage.BindingContext>
```

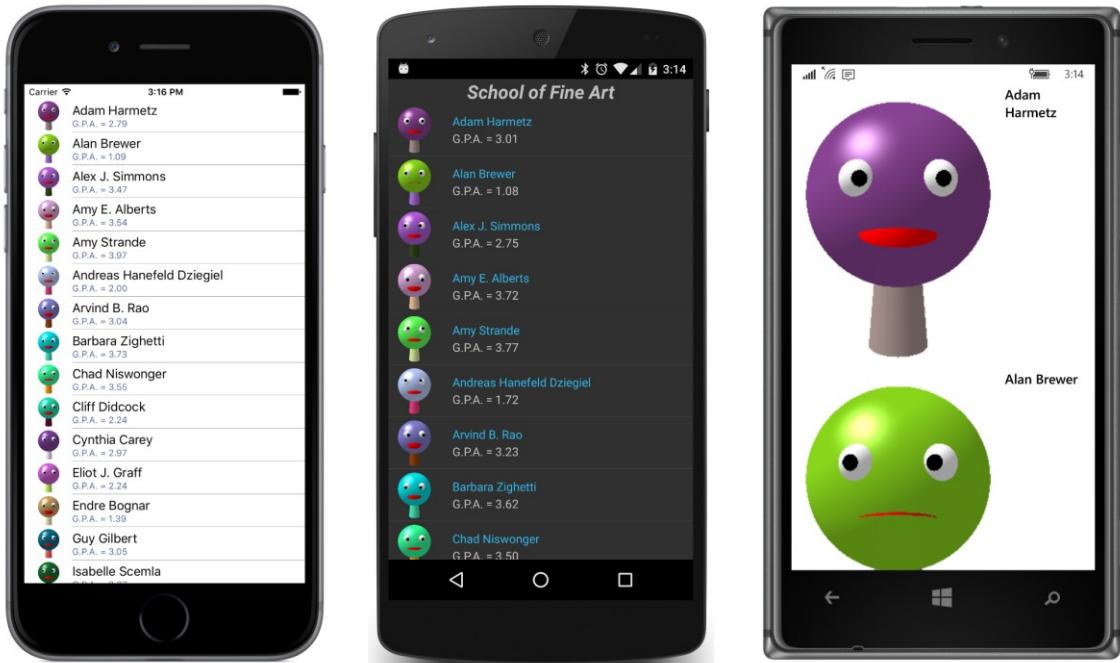
```
< school:SchoolViewModel />
</ ContentPage.BindingContext >

< ListView BindingContext = " {Binding StudentBody} "
    ItemsSource = " {Binding Students} "
    Header = " {Binding School} " >

    < ListView.HeaderTemplate >
        < DataTemplate >
            < Label Text = " {Binding} "
                FontSize = " Large "
                FontAttributes = " Bold, Italic "
                HorizontalTextAlignment = " Center " />
        </ DataTemplate >
    </ ListView.HeaderTemplate >

    < ListView.ItemTemplate >
        < DataTemplate >
            < ImageCell ImageSource = " {Binding PhotoFilename} "
                Text = " {Binding FullName} "
                Detail = " {Binding GradePointAverage,
                    StringFormat='G.P.A. = {0:F2}' } " />
        </ DataTemplate >
    </ ListView.ItemTemplate >
</ ListView >
</ ContentPage >
```

The header shows up only on the Android platform:



Selection and the binding context

The `StudentBody` class doesn't have a property for the selected student. If it did, you could create a data binding between the `SelectedItem` property of the `ListView` and that `selected-student` property in `StudentBody`. As usual with MVVM, the property of the view is the data-binding target and the property in the `ViewModel` is the data-binding source.

However, if you want a detailed view of a student directly, without the intermediary of a `ViewModel`, then the `SelectedItem` property of the `ListView` can be the binding source. The `SelectedStudentDetail` program shows how this might be done. The `ListView` now shares the screen with a

`StackLayout` that contains the detail view. To accommodate landscape and portrait orientations, the `ListView` and `StackLayout` are children of a `Grid` that is manipulated in the code-behind file. The code-behind file also sets the `BindingContext` of the page to an instance of the `SchoolViewModel` class.

The `BindingContext` of the `StackLayout` named "detailLayout" is bound to the `SelectedItem` property of the `ListView`. Because the `SelectedItem` property is of type `Student`, bindings within the `StackLayout` can simply refer to properties of the `Student` class:

```
<ContentPage xmlns = "http://xamarin.com/schemas/2014/forms"
             xmlns:x = "http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class = "SelectedStudentDetail.SelectedStudentDetailPage"
             SizeChanged = "OnPageSizeChanged" >
<ContentPage.Padding >
```

```
< OnPlatform x:TypeArguments = "Thickness" 
    iOS = "0, 20, 0, 0" />

</ ContentPage.Padding >

< Grid x:Name = "mainGrid" >
    < Grid.RowDefinitions >
        < RowDefinition Height = "*" />
        < RowDefinition Height = "*" />
    </ Grid.RowDefinitions >

    < Grid.ColumnDefinitions >
        < ColumnDefinition Width = "*" />
        < ColumnDefinition Width = "0" />
    </ Grid.ColumnDefinitions >

    < ListView x:Name = "listView" 
        Grid.Row = "0"
        Grid.Column = "0"
        ItemsSource = "{Binding StudentBody.Students}" >
        < ListView.ItemTemplate >
            < DataTemplate >
                < ImageCell ImageSource = "{Binding PhotoFilename}" 
                    Text = "{Binding FullName}"
                    Detail = "{Binding GradePointAverage,
                        StringFormat='G.P.A. {0:F2}'}" />
            </ DataTemplate >
        </ ListView.ItemTemplate >
    </ ListView >

    < StackLayout x:Name = "detailLayout" 
        Grid.Row = "1"
        Grid.Column = "0"
        BindingContext = "{Binding Source={x:Reference listView},
            Path=SelectedItem}" >
        < StackLayout Orientation = "Horizontal" 
            HorizontalOptions = "Center"
            Spacing = "0" >
            < StackLayout.Resources >
                < ResourceDictionary >
                    < Style TargetType = "Label" >
                        < Setter Property = "FontSize" Value = "Large" />
                        < Setter Property = "FontAttributes" Value = "Bold" />
                    </ Style >
                </ ResourceDictionary >
            </ StackLayout.Resources >
            < Label Text = "{Binding LastName}" />
            < Label Text = "{Binding FirstName, StringFormat='{0}'}" />
            < Label Text = "{Binding MiddleName, StringFormat='{0}'}" />
        </ StackLayout >

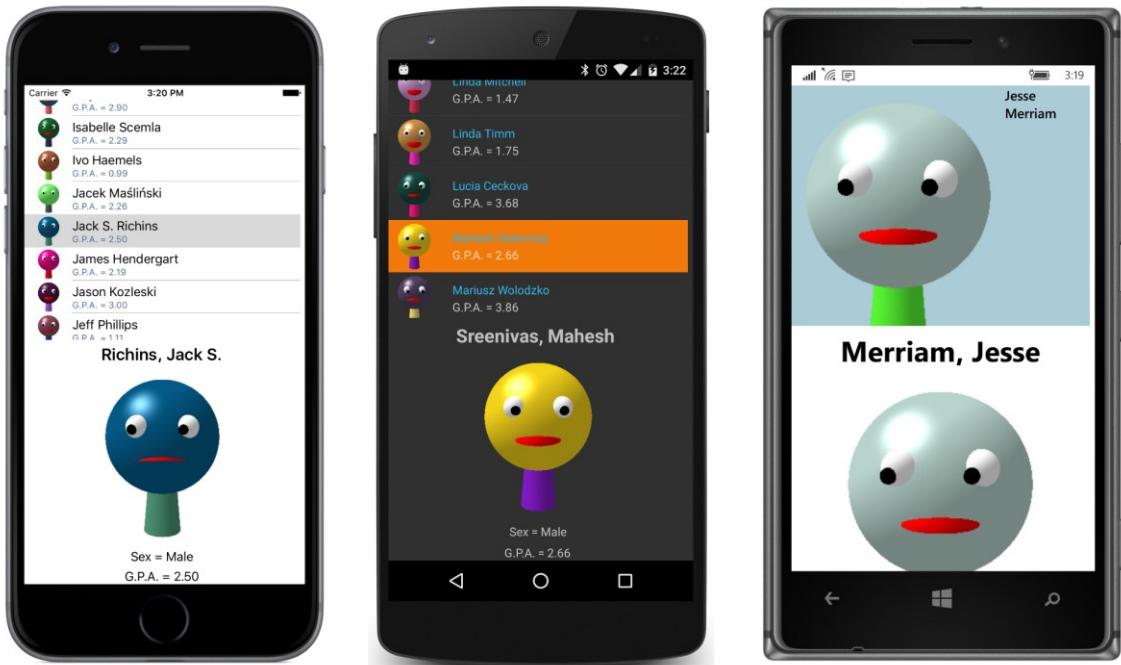
        < Image Source = "{Binding PhotoFilename}" 
            VerticalOptions = "FillAndExpand" />
    </ StackLayout >
```

```

<Label Text = "{Binding Sex, StringFormat='Sex = {0}'}"
       HorizontalOptions = "Center" />
<Label Text = "{Binding GradePointAverage, StringFormat='G.P.A. = {0:F2}'}"
       HorizontalOptions = "Center" />
</StackLayout>
</Grid>
</ContentPage>

```

When you first run the program, the ListView occupies the top half of the page and the entire bottom half of the page is empty. When you select one of the students, the bottom half displays a different formatting of the name, a larger photo (except on the Windows Phone), and additional information:



Notice that all the Label elements in the StackLayout named "detailLayout" have their Text properties set to bindings of properties of the Student class. For example, here are the three Label elements that display the full name in a horizontal StackLayout:

```

<Label Text = "{Binding LastName}" />
<Label Text = "{Binding FirstName, StringFormat='{0}'}" />
<Label Text = "{Binding MiddleName, StringFormat='{0}'}" />

```

An alternative approach is to use separate Label elements for the text that separate the last name and first name and the first name and middle name:

```

<Label Text = "{Binding LastName}" />
<Label Text = ", " />

```

```
<Label Text = "{Binding FirstName}" />
<Label Text = "" />
<Label Text = "{Binding MiddleName}" />
```

Ostensibly, these two approaches seem visually identical. However, if no student is currently selected, the second approach displays a stray comma that looks like an odd speck on the screen. The advantages of using a binding with `StringFormat` is that the Label doesn't appear at all if the `BindingContext` is null.

Sometimes it's unavoidable that some spurious text appears in a detail view when the detail view isn't displaying anything otherwise. In such a case you might want to bind the `IsVisible` property of the detail Layout object to the `SelectedItem` property of the `ListView` with a binding converter that converts null to false and non-null to true.

The code-behind file in the `SelectedStudentDetail` program is responsible for setting the `BindingContext` for the page and also for handling the `SizeChanged` event for the page to adjust the Grid and the `detailLayout` object for a landscape orientation:

```
public partial class SelectedStudentDetailPage : ContentPage
{
    public SelectedStudentDetailPage()
    {
        InitializeComponent();

        // Set BindingContext.
        BindingContext = new SchoolViewModel();
    }

    void OnPageSizeChanged( object sender, EventArgs args )
    {
        // Portrait mode.
        if (Width < Height)
        {
            mainGrid.ColumnDefinitions[0].Width = new GridLength (1, GridUnitType.Star);
            mainGrid.ColumnDefinitions[1].Width = new GridLength (0);

            mainGrid.RowDefinitions[0].Height = new GridLength (1, GridUnitType.Star);
            mainGrid.RowDefinitions[1].Height = new GridLength (1, GridUnitType.Star);

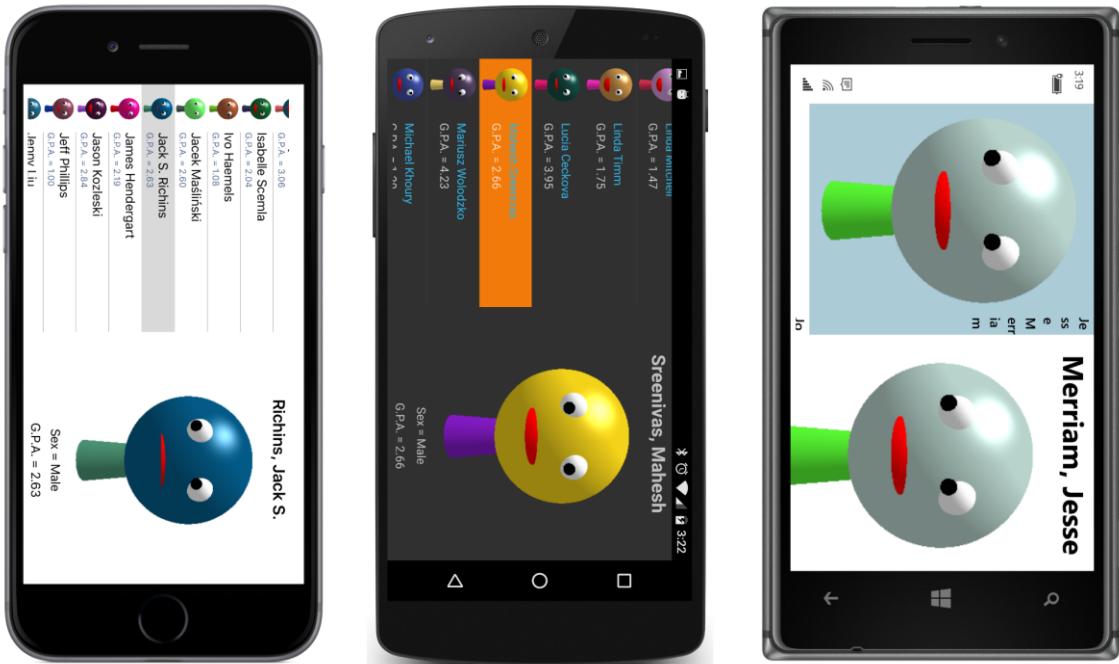
            Grid.SetRow(detailLayout, 1);
            Grid.SetColumn(detailLayout, 0);
        }
        // Landscape mode.
        else
        {
            mainGrid.ColumnDefinitions[0].Width = new GridLength (1, GridUnitType.Star);
            mainGrid.ColumnDefinitions[1].Width = new GridLength (1, GridUnitType.Star);

            mainGrid.RowDefinitions[0].Height = new GridLength (1, GridUnitType.Star);
            mainGrid.RowDefinitions[1].Height = new GridLength (0);

            Grid.SetRow(detailLayout, 0);
        }
    }
}
```

```
        Grid.SetColumn(detailLayout, 1);  
    }  
}
```

Here's a landscape view:



Unfortunately, the large image in the ListView on Windows 10 Mobile crowds out the text.

Dividing a page into a `ListView` and detail view is not the only approach. When the user selects an item in the `ListView`, your program could navigate to a separate page to display the detail view. Or you could make use of a `MasterDetailPage` designed specifically for scenarios such as this. You'll see examples with these solutions in the chapters ahead.

Context menus

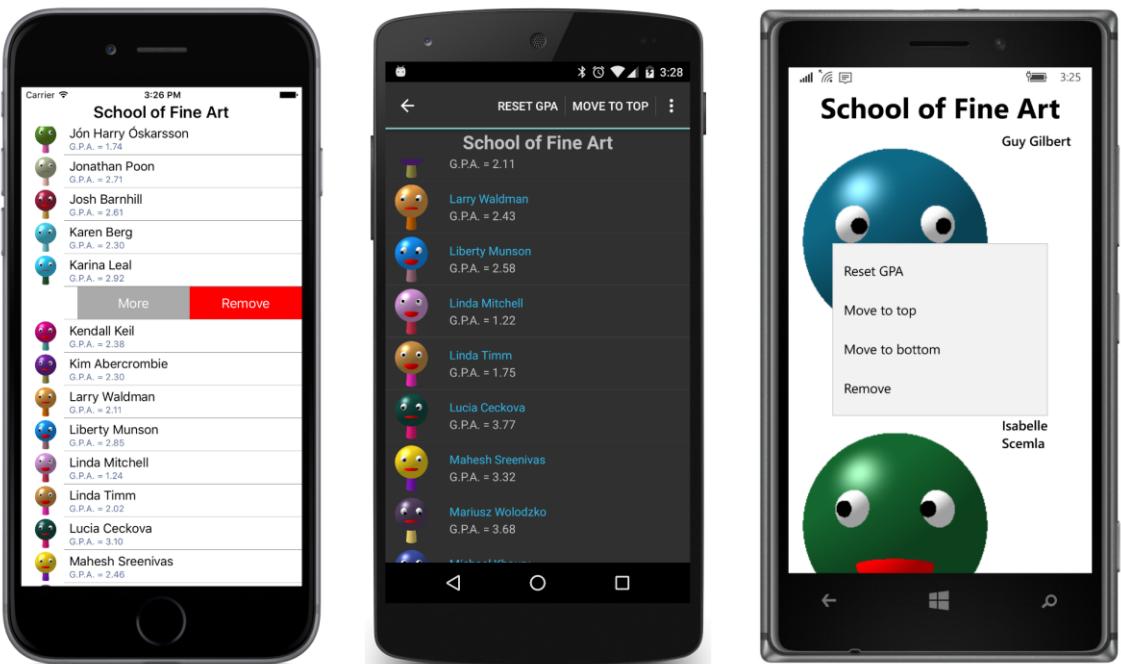
A cell can define a context menu that is invoked in a platform-specific manner. Such a context menu generally allows a user to perform an operation on a specific item in the ListView. When used with a ListView displaying students, for example, such a context menu allows the user to perform actions on a specific student.

The **CellContextMenu** program demonstrates this technique. It defines a context menu with four items:

- **Reset GPA** (which sets the grade point average of the student to 2.5)

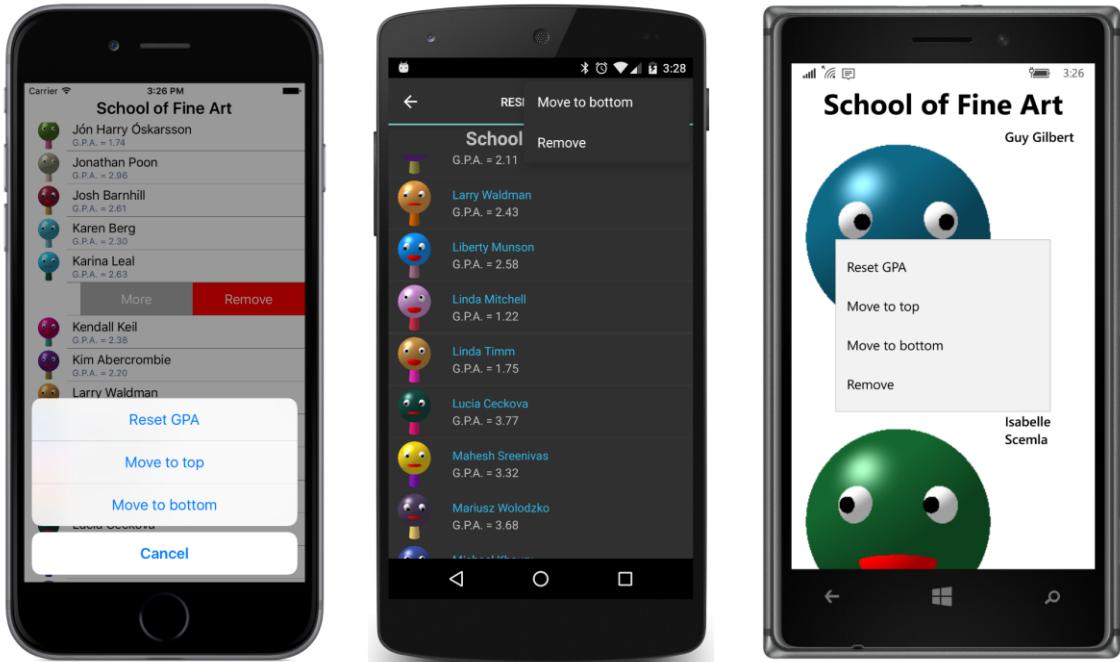
- **Move to Top** (which moves the student to the top of the list)
- **Move to Bottom** (which similarly moves the student to the bottom)
- **Remove** (which removes the student from the list)

On iOS, the context menu is invoked by sliding the item to the left. On Android and Windows 10 Mobile, you press your finger to the item and hold it until the menu appears. Here's the result:



Only one menu item appears on the iOS screen, and that's the item that removes the student from the list. A menu item that removes an entry from the ListView must be specially flagged for iOS. The Android screen lists the first two menu items at the top of the screen. Only the Windows Runtime lists them all.

To see the other menu items, you tap the **More** button on iOS and the vertical ellipsis on Android. The other items appear in a list at the bottom of the iOS screen and in a drop-down list at the top right of the Android screen:



Tapping one of the menu items carries out that operation.

To create a context menu for a cell, you add objects of type `MenuItem` to the `ContextActions` collection defined by the `Cell` class. You've already encountered `MenuItem`. It is the base class for the `ToolbarItem` class described in Chapter 13, "Bitmaps."

`MenuItem` defines five properties:

- Text of type string
- Icon of type `FileImageSource` to access a bitmap from a platform project
- `IsDestructive` of type bool
- Command of type `ICommand`
- `CommandParameter` of type object

In addition, `MenuItem` defines a `Clicked` event. You can handle menu actions either in a `Clicked` handler or—if the menu actions are implemented in a `ViewModel`—an `ICommand` object.

Here's how the `ContextActions` collection is initialized in the `CellContextMenu` program:

```
<ContentPage xmlns = "http://xamarin.com/schemas/2014/forms"
             xmlns:x = "http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:school = "clr-namespace:SchoolOfFineArt;assembly=SchoolOfFineArt"
             x:Class = "CellContextMenu.CellContextMenuPage" >
```

```
< ContentPage.Padding >
    < OnPlatform x:TypeArguments = "Thickness" 
        iOS = "0, 20, 0, 0" />
</ ContentPage.Padding >

< ContentPage.BindingContext >
    < school:SchoolViewModel />
</ ContentPage.BindingContext >

< StackLayout BindingContext = "{Binding StudentBody}" >
    < Label Text = "{Binding School}" 
        FontSize = "Large" 
        FontAttributes = "Bold" 
        HorizontalTextAlignment = "Center" />

    < ListView ItemsSource = "{Binding Students}" >
        < ListView.ItemTemplate >
            < DataTemplate >
                < ImageCell ImageSource = "{Binding PhotoFilename}" 
                    Text = "{Binding FullName}" 
                    Detail = "{Binding GradePointAverage, 
                        StringFormat='G.P.A. = {0:F2}'}" 
                    ContextActions >
                    < MenuItem Text = "Reset GPA" 
                        Command = "{Binding ResetGpaCommand}" />
                    < MenuItem Text = "Move to top" 
                        Command = "{Binding MoveToTopCommand}" />
                    < MenuItem Text = "Move to bottom" 
                        Command = "{Binding MoveToBottomCommand}" />
                    < MenuItem Text = "Remove" 
                        IsDestructive = "True" 
                        Command = "{Binding RemoveCommand}" />
                </ ImageCell.ContextActions >
            </ ImageCell >
        </ DataTemplate >
    </ ListView.ItemTemplate >
</ ListView >
</ StackLayout >
</ ContentPage >
```

Notice that the **IsDestructive** property is set to True for the **Remove** item. This is the property that causes the item to be displayed in red on the iOS screen, and which by convention deletes the item from the collection.

MenuItem defines an **Icon** property that you can set to a bitmap stored in a platform project (much like the icons used with ToolbarItem), but it works only on Android, and the bitmap replaces the Text description.

The **Command** properties of all four MenuItem objects are bound to properties in the **Student** class.

A **Student** object is the binding context for the cell, so it's also the binding context for these **MenuItem** objects. Here's how the properties are defined and initialized in **Student**:

```
public class Student : ViewModelBase
{
    ...
    public Student()
    {
        ResetGpaCommand = new Command(() => GradePointAverage = 2.5);
        MoveToTopCommand = new Command(() => StudentBody.MoveStudentToTop(this));
        MoveToBottomCommand = new Command(() => StudentBody.MoveStudentToBottom(this));
        RemoveCommand = new Command(() => StudentBody.RemoveStudent(this));
    }
    ...
    // Properties for implementing commands.
    [ XmlIgnore ]
    public ICommand ResetGpaCommand { private set; get; }

    [ XmlIgnore ]
    public ICommand MoveToTopCommand { private set; get; }

    [ XmlIgnore ]
    public ICommand MoveToBottomCommand { private set; get; }

    [ XmlIgnore ]
    public ICommand RemoveCommand { private set; get; }

    [ XmlIgnore ]
    public StudentBody StudentBody { set; get; }
}
```

Only the **ResetGpaCommand** can be handled entirely within the **Student** class. The other three commands require access to the collection of students in the **StudentBody** class. For that reason, when first loading in the data, the **SchoolViewModel** sets the **StudentBody** property in each **Student** object to the **StudentBody** object with the collection of students. This allows the **Move** and **Remove** commands to be implemented with calls to the following methods in **StudentBody**:

```
public class StudentBody : ViewModelBase
{
    ...
    public void MoveStudentToTop(Student student)
    {
        Students.Move(Students.IndexOf(student), 0);
    }

    public void MoveStudentToBottom(Student student)
    {
        Students.Move(Students.IndexOf(student), Students.Count - 1);
    }

    public void RemoveStudent(Student student)
    {
        Students.Remove(student);
    }
}
```

```

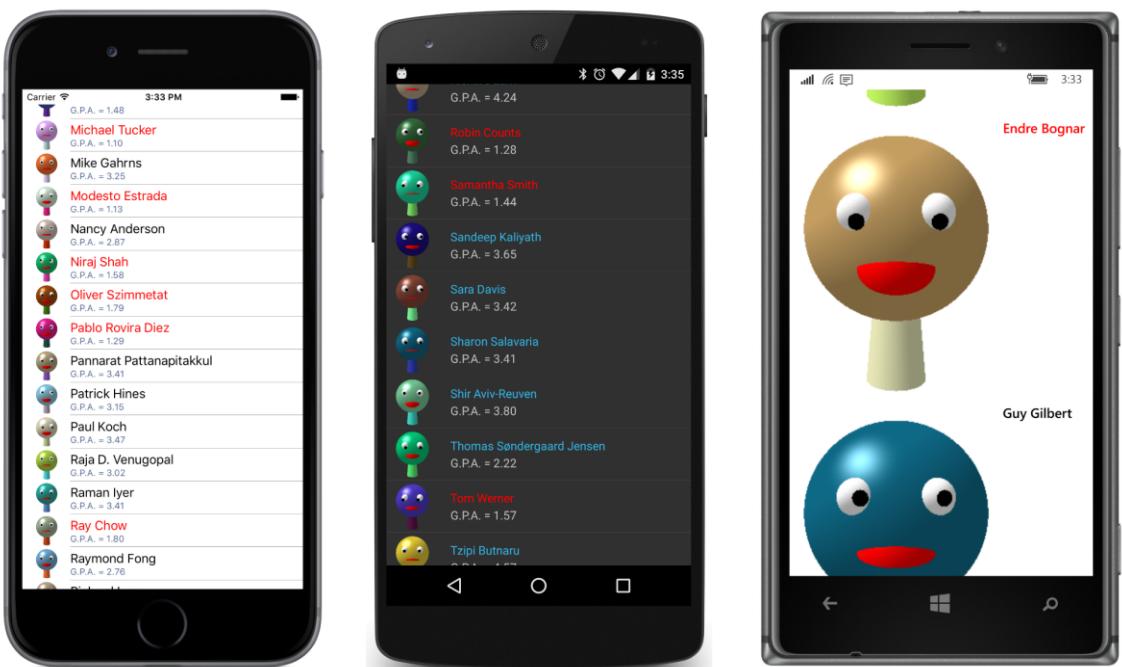
    }
}
}
```

Because the Students collection is an `ObservableCollection`, the `ListView` redraws itself to reflect the new number or new ordering of the students.

Varying the visuals

Sometimes you don't want every item displayed by the `ListView` to be formatted identically. You might want a little different formatting based on the values of some properties. This is generally a job for *triggers*, which you'll be exploring in Chapter 23. However, you can also vary the visuals of items in a `ListView` by using a value converter.

Here's a view of the `ColorCodedStudents` screen. Every student with a grade-point average less than 2.0 is flagged in red, perhaps to highlight the need for some special attention:



In one sense, this is very simple: The `TextColor` property of the `ImageCell` is bound to the `GradePointAverage` property of `Student`. But that's a property of type `Color` bound to a property of type `double`, so a value converter is required, and one that's capable of performing a test on the `GradePointAverage` property to convert to the proper color.

Here is the `ThresholdToObjectConverter` in the `Xamarin.FormsBook.Toolkit` library:

```

namespace Xamarin.FormsBook.Toolkit
{

```

```

public class ThresholdToObjectConverter <T> : IValueConverter
{
    public T TrueObject { set; get; }

    public T FalseObject { set; get; }

    public object Convert( object value, Type targetType,
                          object parameter, CultureInfo culture)
    {
        // Code assumes that all input is valid!
        double number = ( double )value;
        string arg = parameter as string ;
        char op = arg[0];
        double criterion = Double .Parse(arg.Substring(1).Trim());

        switch (op)
        {
            case '<': return number < criterion ? TrueObject : FalseObject;
            case '>': return number > criterion ? TrueObject : FalseObject;
            case '=': return number == criterion ? TrueObject : FalseObject;
        }
        return FalseObject;
    }

    public object ConvertBack( object value, Type targetType,
                             object parameter, CultureInfo culture)
    {
        return 0;
    }
}

```

Like the BoolToObjectConverter described in Chapter 16, “Data binding,” the ThresholdToObjectConverter is a generic class that defines two properties of type T, named TrueObject and FalseObject. But the choice is based on a comparison of the value argument (which is assumed to be of type double) and the parameter argument, which is specified as the ConverterParameter in the binding. This parameter argument is assumed to be a string that contains a one-character comparison operator and a number. For purposes of simplicity and clarity, there is no input validation.

Once the value converter is created, the markup is fairly easy:

```

<ContentPage xmlns = " http://xamarin.com/schemas/2014/forms "
              xmlns:x = " http://schemas.microsoft.com/winfx/2009/xaml "
              xmlns:school = " clr-namespace:SchoolOfFineArt;assembly=SchoolOfFineArt "
              xmlns:toolkit =
                " clr-namespace:Xamarin.FormsBook.Toolkit;assembly=Xamarin.FormsBook.Toolkit "
              x:Class = " ColorCodedStudents.ColorCodedStudentsPage " >

<ContentPage.Padding >
    <OnPlatform x:TypeArguments = " Thickness "
                iOS = " 0, 20, 0, 0 " />
</ContentPage.Padding >

<ContentPage.Resources >

```

```

<ResourceDictionary>
    <toolkit:ThresholdToObjectConverter x:Key = "thresholdConverter"
        x:TypeArguments = "Color"
        TrueObject = "Default"
        FalseObject = "Red" />
</ResourceDictionary>
</ContentPage.Resources>

<ContentPage.BindingContext>
    <school:SchoolViewModel />
</ContentPage.BindingContext>

<ListView ItemsSource = "{Binding StudentBody.Students}" >
    <ListView.ItemTemplate>
        <DataTemplate>
            <ImageCell ImageSource = "{Binding PhotoFilename}"
                Text = "{Binding FullName}"
                TextColor = "{Binding GradePointAverage,
                    Converter={StaticResource thresholdConverter},
                    ConverterParameter=>2}"
                Detail = "{Binding GradePointAverage,
                    StringFormat='G.P.A. = {0:F2}'}" />
        </DataTemplate>
    </ListView.ItemTemplate>
</ListView>
</ContentPage>

```

When the GPA is greater than or equal to 2, the text is displayed in its default color; otherwise the text is displayed in red.

Refreshing the content

As you've seen, if you use an `ObservableCollection` as a source for `ListView`, any change to the collection causes `ObservableCollection` to fire a `CollectionChanged` event and the `ListView` responds by refreshing the display of items.

Sometimes this type of refreshing must be supplemented with something controlled by the user. For example, consider an email client or RSS reader. Such an application might be configured to look for new email or an update to the RSS file every 15 minutes or so, but the user might be somewhat impatient and might want the program to check right away for new data.

For this purpose a convention has developed that is supported by `ListView`. If the `ListView` has its `IsPullToRefresh` property set to true, and if the user swipes down on the `ListView`, the `ListView` will respond by calling the `Execute` method of the `ICommand` object bound to its `RefreshCommand` property. The `ListView` will also set its `IsRefreshing` property to true and display some kind of animation indicating that it's busy.

In reality, the `ListView` is not busy. It's just waiting to be notified that new data is available. You've probably written the code invoked by the `Execute` method of the `ICommand` object to perform an asynchronous operation such as a web access. It must notify the `ListView` that it's finished by setting

the `IsRefreshing` property of the `ListView` back to `false`. At that time, the `ListView` displays the new data and the refresh is complete.

This sounds somewhat complicated, but it gets a lot easier if you build this feature into the `ViewModel` that supplies the data. The whole process is demonstrated with a program called **RssFeed** that accesses an RSS feed from NASA.

The `RssFeedViewModel` class is responsible for downloading the XML with the RSS feed and parsing it. This first happens when the `Url` property is set and the `set` accessor calls the `LoadRssFeed` method:

```
public class RssFeedViewModel : ViewModelBase
{
    string url, title;
    IList < RssItemViewModel > items;
    bool isRefreshing = true;

    public RssFeedViewModel()
    {
        RefreshCommand = new Command (
            execute: () =>
            {
                LoadRssFeed(url);
            },
            canExecute: () =>
            {
                return !isRefreshing;
            });
    }

    public string Url
    {
        set
        {
            if (SetProperty( ref url, value ) && ! String.IsNullOrEmpty(url))
            {
                LoadRssFeed(url);
            }
        }
        get
        {
            return url;
        }
    }

    public string Title
    {
        set { SetProperty( ref title, value ); }
        get { return title; }
    }

    public IList < RssItemViewModel > Items
    {
        set { SetProperty( ref items, value ); }
    }
}
```

```

        get { return items; }
    }

    public ICommand RefreshCommand { private set; get; }

    public bool IsRefreshing
    {
        set { SetProperty(ref isRefreshing, value); }
        get { return isRefreshing; }
    }

    public void LoadRssFeed( string url )
    {
        WebRequest request = WebRequest.Create(url);
        request.BeginGetResponse((args) =>
        {
            // Download XML.
            Stream stream = request.EndGetResponse(args).GetResponseStream();
            StreamReader reader = new StreamReader (stream);
            string xml = reader.ReadToEnd();

            // Parse XML to extract data from RSS feed.
            XDocument doc = XDocument.Parse(xml);
            XElement rss = doc.Element( XName.Get( "rss" ) );
            XElement channel = rss.Element( XName.Get( "channel" ) );

            // Set Title property.
            Title = channel.Element( XName.Get( "title" ) ).Value;

            // Set Items property.
            List < RssItemViewModel > list =
                channel.Elements( XName.Get( "item" ) ).Select(( XElement element ) =>
            {
                // Instantiate RssItemViewModel for each item.
                return new RssItemViewModel (element);
            }).ToList();
            Items = list;
        });
    }
}

```

The LoadRssFeed method uses the LINQ-to-XML interface in the System.Xml.Linq namespace to parse the XML file and set both the Title property and the Items property of the class. The Items property is a collection of RssItemViewModel objects that define five properties associated with each item in the RSS feed. For each item element in the XML file, the LoadRssFeed method instantiates an RssItemViewModel object:

```

public class RssItemViewModel
{
    public RssItemViewModel( XElement element )
    {

```

```

{
    // Although this code might appear to be generalized, it is
    // actually based on desired elements from the particular
    // RSS feed set in the RssFeedPage.xaml file.

    Title = element.Element( XName .Get( "title" )).Value;
    Description = element.Element( XName .Get( "description" )).Value;
    Link = element.Element( XName .Get( "link" )).Value;
    PubDate = element.Element( XName .Get( "pubDate" )).Value;

    // Sometimes there's no thumbnail, so check for its presence.
    XElement thumbnailElement = element.Element(
        XName .Get( "thumbnail" , "http://search.yahoo.com/mrss" ));

    if (thumbnailElement != null )
    {
        Thumbnail = thumbnailElement.Attribute( XName .Get( "url" )).Value;
    }
}

public string Title { protected set ; get ; }

public string Description { protected set ; get ; }

public string Link { protected set ; get ; }

public string PubDate { protected set ; get ; }

public string Thumbnail { protected set ; get ; }
}

```

The constructor of RssFeedViewModel also sets its RefreshCommand property equal to a Command object with an Execute method that also calls LoadRssFeed, which finishes by setting the IsRefreshing property of the class to false. To avoid overlapping web accesses, the CanExecute method of RefreshCommand returns true only if IsRefreshing is false.

Notice that it's not necessary for the Items property in RssFeedViewModel to be an ObservableCollection because once the Items collection is created, the items in the collection never change. When the LoadRssFeed method gets new data, it creates a whole new List object that it sets to the Items property, which results in the firing of a PropertyChanged event.

The RssFeedPage class shown below instantiates the RssFeedViewModel and assigns the Url property. This object becomes the BindingContext for a StackLayout that contains a Label to display the Title property and a ListView. The ItemSource, RefreshCommand, and IsRefreshing properties of the ListView are all bound to properties in the RssFeedViewModel:

```

<ContentPage xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns:x = " http://schemas.microsoft.com/winfx/2009/xaml "
    xmlns:local = "clr-namespace:RssFeed"
    x:Class = "RssFeed.RssFeedPage" >

<ContentPage.Padding >
    <OnPlatform x:TypeArguments = "Thickness"
        iOS = "10, 20, 10, 0" 

```

```
        Android = "10, 0"
        WinPhone = "10, 0" />

    </ContentPage.Padding >

    <ContentPage.Resources>
        <ResourceDictionary>
            <local:RssFeedViewModel x:Key = "rssFeed" 
                Url = "http://earthobservatory.nasa.gov/Feeds/rss/eo_iotd.rss" />
        </ResourceDictionary>
    </ContentPage.Resources>

    <Grid>
        <StackLayout x:Name = "rssLayout" 
            BindingContext = "{StaticResource rssFeed}" >

            <Label Text = "{Binding Title}" 
                FontAttributes = "Bold" 
                HorizontalTextAlignment = "Center" />

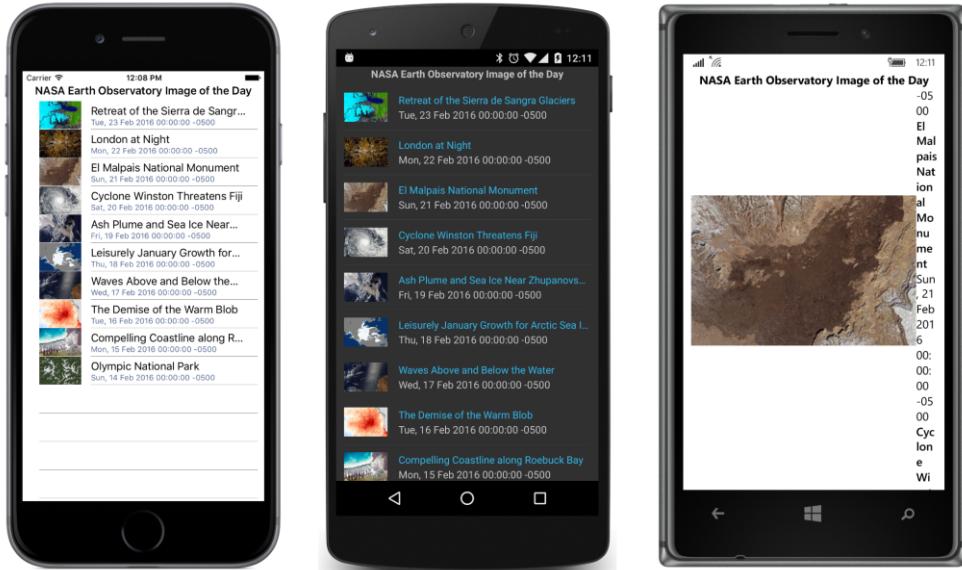
            <ListView x:Name = "listView" 
                ItemsSource = "{Binding Items}" 
                ItemSelected = "OnListViewItemSelected" 
                IsPullToRefreshEnabled = "True" 
                RefreshCommand = "{Binding RefreshCommand}" 
                IsRefreshing = "{Binding IsRefreshing}" >
                <ListView.ItemTemplate>
                    <DataTemplate>
                        <ImageCell Text = "{Binding Title}" 
                            Detail = "{Binding PubDate}" 
                            ImageSource = "{Binding Thumbnail}" />
                    </DataTemplate>
                </ListView.ItemTemplate>
            </ListView>
        </StackLayout >

        <StackLayout x:Name = "webView" 
            IsVisible = "False" >

            <WebView x:Name = "webView" 
                VerticalOptions = "FillAndExpand" />

            <Button Text = "&lt; Back to List" 
                HorizontalOptions = "Center" 
                Clicked = "OnBackButtonClicked" />
        </StackLayout >
    </Grid>
</ContentPage>
```

The items are ideally suited for an ImageCell, but perhaps not on the Windows 10 Mobile device:



When you swipe your finger down this list, the ListView will go into refresh mode by calling the Execute method of the RefreshCommand object and displaying an animation indicating that it's busy. When the IsRefreshing property is set back to false by RssFeedViewModel, the ListView displays the new data. (This is not implemented on the Windows Runtime platforms.)

In addition, the page contains another StackLayout toward the bottom of the XAML file that has its IsVisible property set to false. The first StackLayout with the ListView and this second, hidden StackLayout share a single-cell Grid, so they both essentially occupy the entire page.

When the user selects an item in the ListView, the ItemSelected event handler in the code-behind file hides the StackLayout with the ListView and makes the second StackLayout visible:

```
public partial class RssFeedPage : ContentPage
{
    public RssFeedPage()
    {
        InitializeComponent();
    }

    void OnListViewItemSelected( object sender, SelectedItemChangedEventArgs args )
    {
        if (args.SelectedItem != null)
        {
            // Deselect item.
            (( ListView )sender).SelectedItem = null;

            // Set WebView source to RSS item
            RssItemViewModel rssItem = ( RssItemViewModel )args.SelectedItem;
        }
    }
}
```

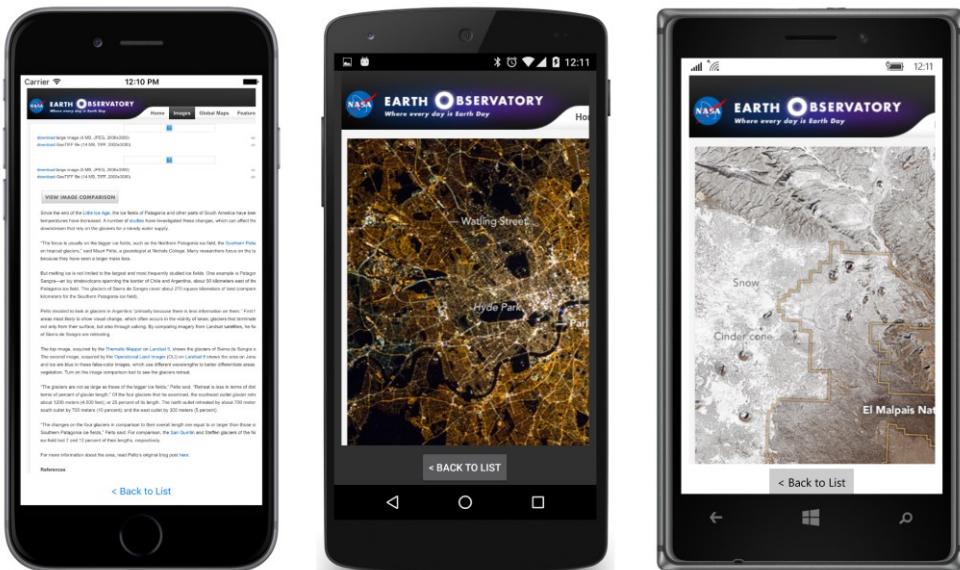
```
// Info.plist to allow accesses to EarthObservatory.nasa.gov sites.
webView.Source = rssItem.Link;

// Hide and make visible.
rssLayout.isVisible = false;
webLayout.isVisible = true;
}

}

void OnBackButtonClicked(object sender, EventArgs args)
{
    // Hide and make visible.
    webLayout.isVisible = false;
    rssLayout.isVisible = true;
}
}
```

This second StackLayout contains a WebView for a display of the item referenced by the RSS feed item and a button to go back to the ListView:



Notice how the ItemSelected event handler sets the SelectedItem property of the ListView to null, effectively deselecting the item. (However, the selected item is still available in the SelectedItem property of the event arguments.) This is a common technique when using the ListView for navigational purposes. When the user returns to the ListView, you don't want the item to be still selected. Setting the SelectedItem property of the ListView to null causes another call to the ItemSelected event handler, of course, but if the handler begins by ignoring cases when SelectedItem is null, the second call shouldn't be a problem.

A more sophisticated program would navigate to a second page or use the detail part of a MasterDetailPage for displaying the item. Those techniques will be demonstrated in future chapters.

The TableView and its intents

The third of the three collection views in Xamarin.Forms is TableView, and the name might be a little deceptive. When we hear the word “table” in programming contexts, we usually think of a two-dimensional grid, such as an HTML table. The **Xamarin.Forms TableView is instead a vertical, scrollable list of items that are visually generated from Cell classes**. This might sound very similar to a ListView, but the ListView and TableView are quite different in use:

The ListView generally displays a list of items of the same type, usually instances of a particular data class. These items are in an **IEnumerable** collection. The ListView specifies a single Cell derivative for rendering these data objects. Items are selectable.

The TableView displays a list of items of different types. In real-life programming, often these items are properties of a single class. Each item is associated with its own Cell to display the property and often to allow the user to interact with the property. In the general case, the TableView displays more than one type of cell.

Properties and hierarchies

ListView and ItemsView together define 18 properties, while TableView has only four:

- Intent of type TableIntent.
- Root of type TableRoot. (This is the content property of TableView.)
- RowHeight of type int.
- HasUnevenRows of type bool.

The RowHeight and HasUnevenRows properties play the same role in the TableView as in the ListView.

Perhaps the most revealing property of the TableView class is a property that is *not* guaranteed to have any effect on functionality and appearance. This property is named Intent, and it indicates how you’re using the particular TableView in your program. You can set this property (or not) to a member of the TableIntent enumeration:

- Data
- Form
- Settings

- Menu

These members suggest the various ways that you can use TableView. When used for Data, the TableView usually displays related items, but items of different types. A Form is a series of items that the user interacts with to enter information. A TableView used for program Settings is sometimes known as a *dialog*. This use is similar to Form, except that settings usually have default values. You can also use a TableView for a Menu, in which case the items are generally displayed using text or bitmaps and initiate an action when tapped.

The Root property defines the root of the hierarchy of items displayed by the TableView. Each item in a TableView is associated with a single Cell derivative, and the various cells can be organized into sections. To support this hierarchy of items, several classes are defined:

- TableSectionBase is an abstract class that derives from BindableObject and defines a Title property.
- TableSectionBase<T> is an abstract class that derives from TableSectionBase and implements the IList<T> interface, and hence also the ICollection<T> and IEnumerable<T> interfaces. The class also implements the INotifyCollectionChanged interface; internally it maintains an ObservableCollection<T> for this collection. This allows items to be dynamically added to or removed from the TableView.
- TableSection derives from TableSectionBase<Cell>.
- TableRoot derives from TableSectionBase<TableSection>.

In summary, TableView has a Root property that you set to a TableRoot object, which is a collection of TableSection objects, each of which is a collection of Cell objects.

Notice that both TableSection and TableRoot inherit a Title property from TableSectionBase. Depending on the derived class, this is either a title for the section or a title for the entire table. Both TableSection and TableRoot have constructors that let you set this Title property when creating the object.

The TableSectionBase<T> class defines two Add methods for adding items to the collection. The first Add method is required by the ICollection interface; the second is not:

- public void Add(T item)
- public void Add(IEnumerable<T> items)

This second Add method seems to allow you to add one TableSection to another TableSection, and one TableRoot to another TableRoot, and that process might seem to imply that you can have a nested series of TableRoot or TableSection instances. But that is not so. This Add method just transfers the items from one collection to another. The hierarchy never gets any deeper than a TableRoot that is a collection of TableSection objects, which are collections of Cell objects.

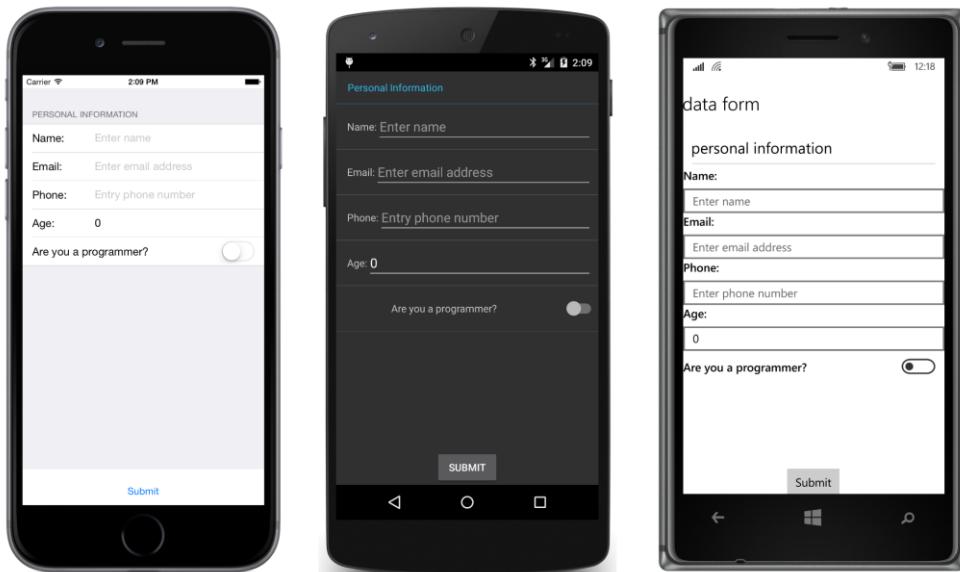
Although the TableView makes use of Cell objects, it does not use DataTemplate. Whether you

define a `TableView` in code or in XAML, you always set data bindings directly on the `Cell` objects. Generally these bindings are very simple because you set a `BindingContext` on the `TableView` that is inherited by the individual items.

Visually and functionally, the `TableView` is not very different from a `StackLayout` in a `ScrollView`, where the `StackLayout` contains a collection of short visual trees with bindings. But generally the `TableView` is more convenient in organizing and arranging the information.

A prosaic form

Let's make a data-entry form that lets the program's user enter a person's name and some other information. When you first run the **EntryForm** program, it looks like this:



The `TableView` consists of everything on the page except the `Submit` button. This `TableView` has one `TableSection` consisting of five cells—four `EntryCell` elements and one `SwitchCell`. (Those are the only two `Cell` derivatives you haven't seen yet.) The text "Data Form" is the `Title` property of the `TableRoot` object, and it shows up only on the Windows 10 Mobile screen. The text "Personal Information" is the `Title` property for the `TableSection`.

The five cells correspond to five properties of this little class named `PersonalInformation`. Although the class name doesn't explicitly identify this as a `ViewModel`, the class derives from `ViewModelBase`:

```
class PersonalInformation : ViewModelBase
{
    string name, emailAddress, phoneNumber;
    int age;
```

```
bool isProgrammer;

public string Name
{
    set { SetProperty( ref name, value ); }
    get { return name; }
}

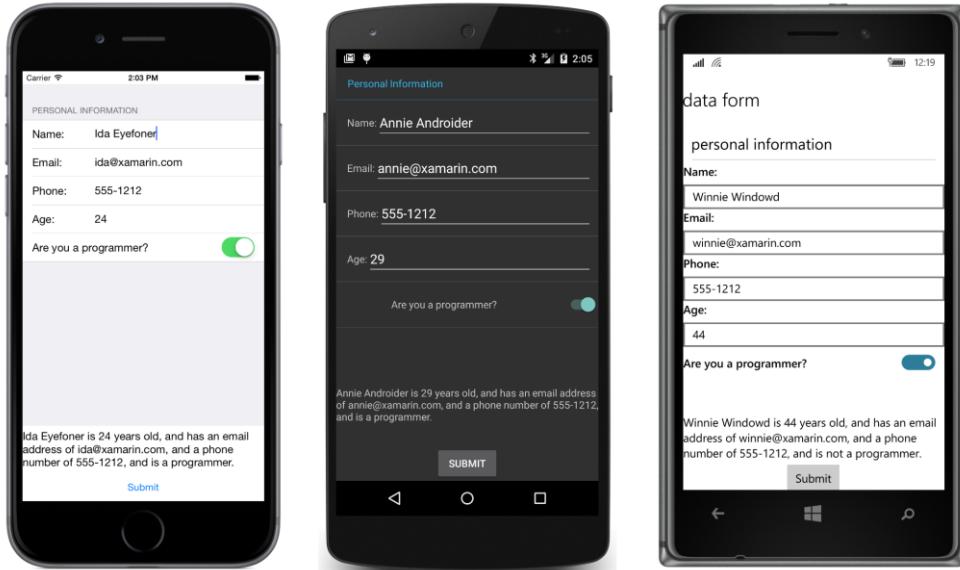
public string EmailAddress
{
    set { SetProperty( ref emailAddress, value ); }
    get { return emailAddress; }
}

public string PhoneNumber
{
    set { SetProperty( ref phoneNumber, value ); }
    get { return phoneNumber; }
}

public int Age
{
    set { SetProperty( ref age, value ); }
    get { return age; }
}

public bool IsProgrammer
{
    set { SetProperty( ref isProgrammer, value ); }
    get { return isProgrammer; }
}
```

When you fill in the information in the form and press the **Submit** button, the program displays the information from the `PersonalInformation` instance in a little paragraph at the bottom of the screen:



This program maintains just a single instance of `PersonalInformation`. A real application would perhaps create a new instance for each person whose information the user is supplying, and then store each instance in an `ObservableCollection<PersonalInformation>` for display by a `ListView`.

The EntryForm XAML file instantiates PersonalInformation as the BindingContext of the TableView. You can see here the TableRoot, the TableSection, and the five Cell objects:

```
<ContentPage xmlns = "http://xamarin.com/schemas/2014/forms"
    xmlns:x = "http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:local = "clr-namespace:EntryForm"
    x:Class = "EntryForm.EntryFormPage" >

<ContentPage.Padding>
    <OnPlatform x:TypeArguments = "Thickness"
        iOS = "0, 20, 0, 0" />
</ContentPage.Padding>

<StackLayout>
    <TableView x:Name = "tableView"
        Intent = "Form" >

        <TableView.BindingContext>
            <local:PersonalInformation />
        </TableView.BindingContext >

        <TableRoot Title = "Data Form" >
            <TableSection Title = "Personal Information" >
                <EntryCell Label = "Name:" >
                    Text = "{Binding Name}"
                    Placeholder = "Enter name"
                    Keyboard = "Text" />
                </EntryCell>
            </TableSection>
        </TableRoot>
    </TableView>
</StackLayout>
```

```

<EntryCell Label = "Email:">
    Text = "{Binding EmailAddress}"
    Placeholder = "Enter email address"
    Keyboard = "Email" />

<EntryCell Label = "Phone:">
    Text = "{Binding PhoneNumber}"
    Placeholder = "Enter phone number"
    Keyboard = "Telephone" />

<EntryCell Label = "Age:">
    Text = "{Binding Age}"
    Placeholder = "Enter age"
    Keyboard = "Numeric" />

<SwitchCell Text = "Are you a programmer?">
    On = "{Binding IsProgrammer}" />
</TableSection>
</TableRoot>
</TableView>

<Label x:Name = "summaryLabel">
    VerticalOptions = "CenterAndExpand" />

<Button Text = "Submit">
    HorizontalOptions = "Center"
    Clicked = "OnSubmitButtonClicked" />
</StackLayout>
</ContentPage>

```

Each of the properties of the `PersonalInformation` class corresponds to a Cell. For four of these properties, this is an `EntryCell` that consists (at least conceptually) of an identifying Label and an `Entry` view. (In reality, the `EntryCell` consists of platform-specific visual objects, but it's convenient to speak of these objects using `Xamarin.Forms` names.) The `Label` property specifies the text that appears at the left; the `Placeholder` and `Keyboard` properties of `EntryView` duplicate the same properties in `Entry`. A `Text` property indicates the text in the `Entry` view.

The fifth cell is a `SwitchCell` for the Boolean property `IsProgrammer`. In this case, the `Text` property specifies the text at the left of the cell, and the `On` property indicates the state of the Switch.

Because the `BindingContext` of the `TableView` is `PersonalInformation`, the bindings in the Cell objects can simply reference the properties of `PersonalInformation`. The binding modes of the `Text` property of the `EntryCell` and the `On` property of the `SwitchCell` are both `TwoWay`. If you only need to transfer data from the view to the data class, this mode can be `OneWayToSource`, but in general you might want to initialize the views from the data class. For example, you can instantiate the `PersonalInformation` instance in the XAML file like this:

```

<TableView.BindingContext>
    <local:PersonalInformation Name = "Naomi Name">
        EmailAddress = "naomi@xamarin.com"
        PhoneNumber = "555-1212"
    </local:PersonalInformation>
</TableView.BindingContext>

```

```

Age = "29"
IsProgrammer = "True" />
</TableView.BindingContext >
```

The cells will then be initialized with that information when the program starts up.

Both EntryCell and SwitchCell fire events if you prefer obtaining information through event handling rather than data binding.

The code-behind file simply processes the Clicked event of the Submit button by creating a text string with the information from the PersonallInformation instance and displaying it with the Label:

```

public partial class EntryFormPage : ContentPage
{
    public EntryFormPage()
    {
        InitializeComponent();
    }

    void OnSubmitButtonClicked( object sender, EventArgs args )
    {
        PersonallInformation personalInfo = ( PersonallInformation )tableView.BindingContext;

        summaryLabel.Text = String .Format(
            "(0) is {1} years old, and has an email address " +
            "of {2}, and a phone number of {3}, and is {4}" +
            "a programmer.",
            personalInfo.Name, personalInfo.Age,
            personalInfo.EmailAddress, personalInfo.PhoneNumber,
            personalInfo.IsProgrammer ? "" : "not" );
    }
}
```

Custom cells

Of course, few people are entirely happy with the first version of an application, and perhaps that is true for the simple **EntryForm** program. Perhaps the revised design requirements eliminate the integer

Age property from PersonallInformation and substitute a text AgeRange property with some fixed ranges. Two more properties are added to the class that pertain only to programmers: These are properties of type string that indicate the programmer's preferred computer language and platform, choosable from lists of languages and platforms.

Here's the revised ViewModel class, now called ProgrammerInformation:

```

class ProgrammerInformation : ViewModelBase
{
    string name, emailAddress, phoneNumber, ageRange;
    bool isProgrammer;
    string language, platform;
```

```
public string Name
{
    set { SetProperty( ref name, value ); }
    get { return name; }
}

public string EmailAddress
{
    set { SetProperty( ref emailAddress, value ); }
    get { return emailAddress; }
}

public string PhoneNumber
{
    set { SetProperty( ref phoneNumber, value ); }
    get { return phoneNumber; }
}

public string AgeRange
{
    set { SetProperty( ref ageRange, value ); }
    get { return ageRange; }
}

public bool IsProgrammer
{
    set { SetProperty( ref isProgrammer, value ); }
    get { return isProgrammer; }
}

public string Language
{
    set { SetProperty( ref language, value ); }
    get { return language; }
}

public string Platform
{
    set { SetProperty( ref platform, value ); }
    get { return platform; }
}
```

The `AgeRange`, `Language`, and `Platform` properties seem ideally suited for Picker, but using a Picker inside a TableView requires that the Picker be part of a ViewCell. How do we do this?

When working with a ListView, the simplest way to create a custom cell involves defining a visual tree in a ViewCell within a DataTemplate right in XAML. This approach makes sense because the visual tree that you define is probably tailored specifically to the items in the ListView and is probably not going to be reused somewhere else.

You can use that same technique with a TableView, but with a TableView it's more likely that you'll be reusing particular types of interactive cells. For example, the ProgrammerInformation class

has three properties that are suitable for Picker. This implies that it makes more sense to create a custom PickerCell class that you can use here and elsewhere.

The **Xamarin.FormsBook.Toolkit** library contains a PickerCell class that derives from ViewCell and is basically a wrapper around a Picker view. The class consists of a XAML file and a code-behind file. The code-behind file defines three properties backed by bindable properties: Label (which identifies the cell just like the Label property in EntryCell), Title (which corresponds to the Title property of Picker), and SelectedValue, which is the actual string selected in the Picker. In addition, a get-only Items property exposes the Items collection of the Picker:

```
namespace Xamarin.FormsBook.Toolkit
{
    [ContentProperty ("Items")]
    public partial class PickerCell : ViewCell
    {
        public static readonly BindableProperty LabelProperty =
            BindableProperty.Create(
                "Label", typeof (string), typeof (PickerCell), default (string));

        public static readonly BindableProperty TitleProperty =
            BindableProperty.Create(
                "Title", typeof (string), typeof (PickerCell), default (string));

        public static readonly BindableProperty SelectedValueProperty =
            BindableProperty.Create(
                "SelectedValue", typeof (string), typeof (PickerCell), null,
                BindingMode.TwoWay,
                propertyChanged: (sender, oldValue, newValue) =>
                {
                    PickerCell pickerCell = ( PickerCell )sender;

                    if ( String.IsNullOrEmpty(newValue))
                    {
                        pickerCell.picker.SelectedIndex = -1;
                    }
                    else
                    {
                        pickerCell.picker.SelectedIndex =
                            pickerCell.Items.IndexOf(newValue);
                    }
                });
    }

    public PickerCell()
    {
        InitializeComponent();
    }

    public string Label
    {
        set { SetValue(LabelProperty, value); }
        get { return ( string )GetValue(LabelProperty); }
    }
}
```

```

public string Title
{
    get { return ( string )GetValue(TitleProperty); }
    set { SetValue(TitleProperty, value ); }
}

public string SelectedValue
{
    get { return ( string )GetValue(SelectedValueProperty); }
    set { SetValue(SelectedValueProperty, value ); }
}

// Items property.
public IList < string > Items
{
    get { return picker.Items; }
}

void OnPickerSelectedIndexChanged( object sender, EventArgs args )
{
    if (picker.SelectedIndex == -1)
    {
        SelectedValue = null ;
    }
    else
    {
        SelectedValue = Items[picker.SelectedIndex];
    }
}
}

```

The XAML file defines the visual tree of PickerCell, which simply consists of an identifying Label and the Picker itself. Notice that the root element of the XAML file is ViewCell, which is the class that PickerCell derives from:

```
<ViewCell xmlns = "http://xamarin.com/schemas/2014/forms"
          xmlns:x = "http://schemas.microsoft.com/winfx/2009/xaml"
          x:Class = "Xamarin.FormsBook.Toolkit.PickerCell"
          x:Name = "cell">

    <ViewCell.View>
        <StackLayout Orientation = "Horizontal"
                    BindingContext = "{x:Reference cell}"
                    Padding = "16, 0">

            <Label Text = "{Binding Label}"
                  VerticalOptions = "Center"/>

            <Picker x:Name = "picker"
                   Title = "{Binding Title}"
                   VerticalOptions = "Center"
                   HorizontalOptions = "FillAndExpand"
                   SelectedIndexChanged = "OnPickerSelectedIndexChanged"/>
        
```

```
</ StackLayout >
</ ViewCell.View >
</ ViewCell >
```

The Padding value set on the StackLayout was chosen empirically to be visually consistent with the Xamarin.Forms EntryCell.

Normally the ViewCell.View property element tags wouldn't be required in this XAML file because View is the content property of ViewCell. However, the code-behind file defines the content property of PickerCell to be the Items collection, which means that the content property is no longer View and the ViewCell.View tags are necessary.

The root element of the XAML file has an x:Name attribute that gives the object a name of "cell," and the StackLayout sets its BindingContext to that object, which means that the BindingContext for the children of the StackLayout is the PickerCell instance itself. This allows the Label and Picker to contain bindings to the Label and Title properties defined by PickerCell in the code-behind file.

The Picker fires a SelectedIndexChanged event that is handled in the code-behind file so that the code-behind file can convert the SelectedIndex of the Picker to a SelectedValue of the PickerCell.

This is not the only way to create a custom PickerCell class. You can also create it by defining individual PickerCellRenderer classes for each platform.

The TableView in the ConditionalCells program uses this PickerCell for three of the properties in the ProgrammerInformation class and initializes each PickerCell with a collection of strings:

```
< ContentPage xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns:x = " http://schemas.microsoft.com/winfx/2009/xaml "
    xmlns:local = " clr-namespace:ConditionalCells "
    xmlns:toolkit =
        " clr-namespace:Xamarin.FormsBook.Toolkit;assembly=Xamarin.FormsBook.Toolkit "
    x:Class = " ConditionalCells.ConditionalCellsPage " >

< ContentPage.Padding >
    < OnPlatform x:TypeArguments = " Thickness " >
        iOS = " 0, 20, 0, 0 " />
    </ ContentPage.Padding >

    < StackLayout >
        < TableView Intent = " Form " >

            < TableView.BindingContext >
                < local:ProgrammerInformation />
            </ TableView.BindingContext >

            < TableRoot Title = " Data Form " >
                < TableSection Title = " Personal Information " >
                    < EntryCell Label = " Name: " >
                        Text = " {Binding Name} "
                    </ EntryCell >
                </ TableSection >
            </ TableRoot >
        </ TableView >
    </ StackLayout >
</ ContentPage >
```

```
Placeholder = " Enter name "
Keyboard = "Text "/>

<EntryCell Label = " Email: "
Text = "{Binding EmailAddress}"
Placeholder = " Enter email address "
Keyboard = "Email "/>

<EntryCell Label = " Phone: "
Text = "{Binding PhoneNumber}"
Placeholder = " Enter phone number "
Keyboard = "Telephone "/>

<toolkit:PickerCell Label = " Age Range: "
Title = " Age Range "
SelectedValue = "{Binding AgeRange}">
<x:String> 10 - 19 </x:String>
<x:String> 20 - 29 </x:String>
<x:String> 30 - 39 </x:String>
<x:String> 40 - 49 </x:String>
<x:String> 50 - 59 </x:String>
<x:String> 60 - 99 </x:String>
</toolkit:PickerCell >

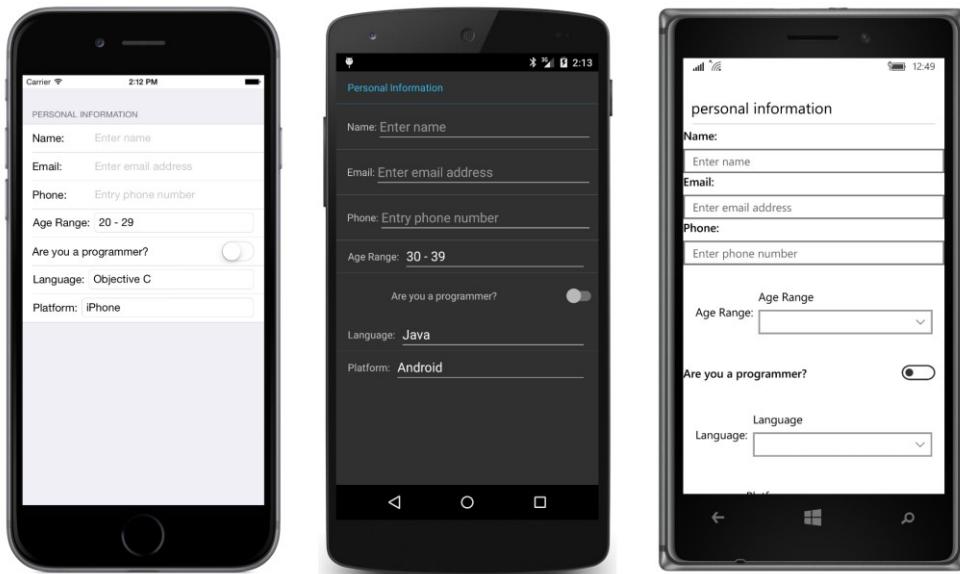
<SwitchCell Text = " Are you a programmer? "
On = "{Binding IsProgrammer}" />

<toolkit:PickerCell Label = " Language: "
Title = " Language "
IsEnabled = "{Binding IsProgrammer}"
SelectedValue = "{Binding Language}">
<x:String> C </x:String>
<x:String> C++ </x:String>
<x:String> C# </x:String>
<x:String> Objective C </x:String>
<x:String> Java </x:String>
<x:String> Other </x:String>
</toolkit:PickerCell >

<toolkit:PickerCell Label = " Platform: "
Title = " Platform "
IsEnabled = "{Binding IsProgrammer}"
SelectedValue = "{Binding Platform}">
<x:String> iPhone </x:String>
<x:String> Android </x:String>
<x:String> Windows Phone </x:String>
<x:String> Other </x:String>
</toolkit:PickerCell >
</TableSection >
</TableRoot >
</TableView >
</StackLayout >
</ContentPage >
```

Notice how the `IsEnabled` properties of the `PickerCell` for both the Platform and Language properties are bound to the `IsProgrammer` property, which means that these cells should be disabled unless the `SwitchCell` is flipped on and the `IsProgrammer` property is true. That's why this program is called **ConditionalCells**.

However, it doesn't seem to work, as this screenshot verifies:



Even though the `IsProgrammer` switch is off, and the `IsEnabled` property of each of the last two `PickerCell` elements is set to false, those elements still respond and allow selecting a value. Moreover, the `PickerCell` doesn't look or work very well on the Windows 10 Mobile platform.

So let's try another approach.

Conditional sections

A `TableView` can have multiple sections, and you might want a section to be entirely invisible if it doesn't currently apply. In the previous example, a second section, titled "Programmer Information," might contain the two `PickerCell` elements for the Language and Platform properties. To make the section visible or hidden, the section can be added to or removed from the `TableRoot` based on the setting of the `IsProgrammer` property. (Recall that the internal collections in `TableView` are of type `ObservableCollection`, so the `TableView` should respond to items added or removed dynamically from these collections.) Unfortunately, this can't be handled entirely in XAML, but the code support is fairly easy.

Here is the XAML file in the **ConditionalSection** program. It is the same as the XAML file in the previous program except that the `BindingContext` is no longer set on the `TableView` (that happens in

the code-behind file) and the last two PickerCell elements have been moved into a second section with the heading "Programmer Information":

```
<ContentPage xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns:x = " http://schemas.microsoft.com/winfx/2009/xaml "
    xmlns:local = " clr-namespace:ConditionalSection "
    xmlns:toolkit =
        " clr-namespace:Xamarin.FormsBook.Toolkit;assembly=Xamarin.FormsBook.Toolkit "
    x:Class = " ConditionalSection.ConditionalSectionPage " >

< ContentPage.Padding >
    < OnPlatform x:TypeArguments = " Thickness " >
        iOS = " 0, 20, 0, 0 " />
</ ContentPage.Padding >

< StackLayout >
    < TableView x:Name = " tableView " >
        Intent = " Form " >
        < TableRoot Title = " Data Form " >
            < TableSection Title = " Personal Information " >
                < EntryCell Label = " Name: " >
                    Text = " {Binding Name} "
                    Placeholder = " Enter name "
                    Keyboard = " Text " />

                < EntryCell Label = " Email: " >
                    Text = " {Binding EmailAddress} "
                    Placeholder = " Enter email address "
                    Keyboard = " Email " />

                < EntryCell Label = " Phone: " >
                    Text = " {Binding PhoneNumber} "
                    Placeholder = " Enter phone number "
                    Keyboard = " Telephone " />

                < toolkit:PickerCell Label = " Age Range: " >
                    Title = " Age Range "
                    SelectedValue = " {Binding AgeRange} " >
                    < x:String > 10 - 19 </ x:String >
                    < x:String > 20 - 29 </ x:String >
                    < x:String > 30 - 39 </ x:String >
                    < x:String > 40 - 49 </ x:String >
                    < x:String > 50 - 59 </ x:String >
                    < x:String > 60 - 99 </ x:String >
                </ toolkit:PickerCell >

                < SwitchCell x:Name = " isProgrammerSwitch " >
                    Text = " Are you a programmer? "
                    On = " {Binding IsProgrammer} " />

            </ TableSection >
            < TableSection x:Name = " programmerInfoSection " >
                Title = " Programmer Information " >
                < toolkit:PickerCell Label = " Language: " >
```

```

        Title = "Language"
        SelectedValue = "{Binding Language}">
<x:String> C </x:String>
<x:String> C++ </x:String>
<x:String> C# </x:String>
<x:String> Objective C </x:String>
<x:String> Java </x:String>
<x:String> Other </x:String>
</ toolkit:PickerCell >

< toolkit:PickerCell Label = "Platform:">
        Title = "Platform"
        SelectedValue = "{Binding Platform}">
<x:String> iPhone </x:String>
<x:String> Android </x:String>
<x:String> Windows Phone </x:String>
<x:String> Other </x:String>
</ toolkit:PickerCell >
</ TableSection >
</ TableRoot >
</ TableView >
</ StackLayout >
</ ContentPage >
```

The constructor in the code-behind file handles the rest. It creates the `ProgrammerInformation` object to set to the `BindingContext` of the `TableView` and then removes the second `TableSection` from the `TableRoot`. The page constructor then sets a handler for the `PropertyChanged` event of `ProgrammerInformation` and waits for changes to the `IsProgrammer` property:

```

public partial class ConditionalSectionPage : ContentPage
{
    public ConditionalSectionPage()
    {
        InitializeComponent();

        // Set BindingContext of TableView.
        ProgrammerInformation programmerInfo = new ProgrammerInformation();
        tableView.BindingContext = programmerInfo;

        // Remove programmer-information section!
        tableView.Root.Remove(programmerInfoSection);

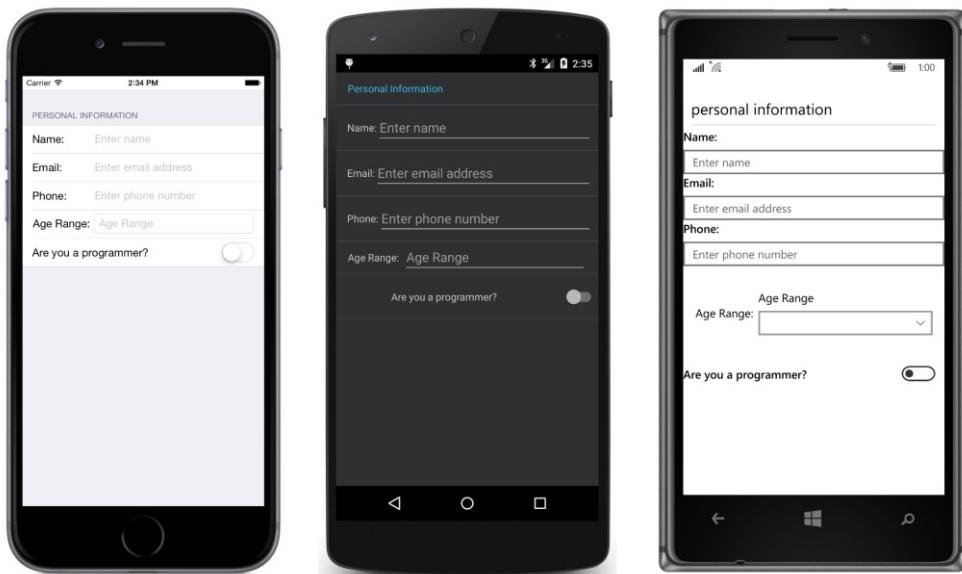
        // Watch for changes in IsProgrammer property in ProgrammerInformation.
        programmerInfo.PropertyChanged += (sender, args) =>
        {
            if (args.PropertyName == "IsProgrammer")
            {
                if (programmerInfo.IsProgrammer &&
                    tableView.Root.IndexOf(programmerInfoSection) == -1)
                {
                    tableView.Root.Add(programmerInfoSection);
                }
                if (!programmerInfo.IsProgrammer &&

```

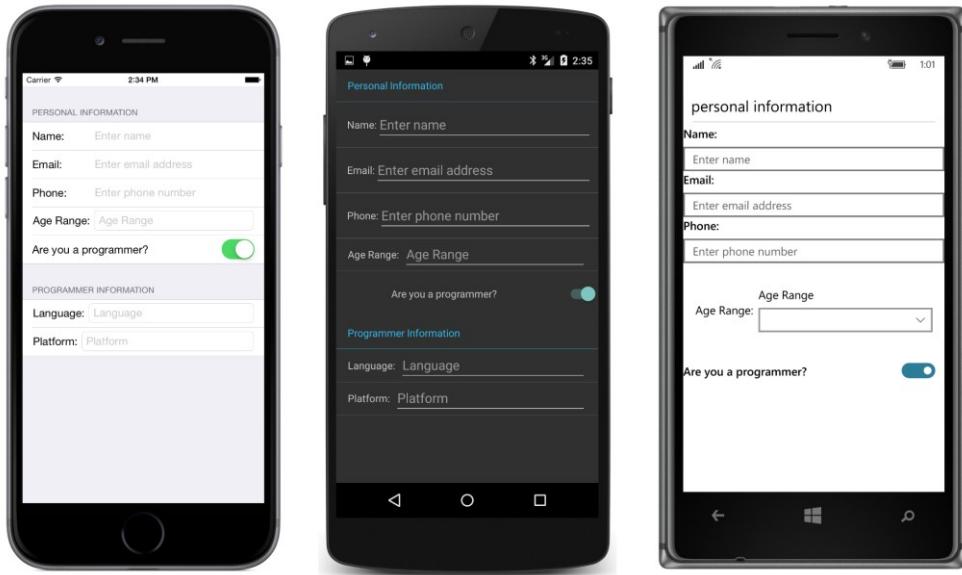
```
        tableView.Root.IndexOf(programmerInfoSection) != -1)
    {
        tableView.Root.Remove(programmerInfoSection);
    }
}
}
}
}
```

In theory, the `PropertyChanged` handler doesn't need to check if the `TableSection` is already part of the `TableRoot` collection before adding it, or check if it's not part of the collection before attempting to remove it, but the checks don't hurt.

Here's the program when it first starts up with only one section visible:



Toggling the `SwitchCell` on brings the two additional properties into view:



But not on the Windows 10 Mobile screen.

You don't need to have a single `BindingContext` for the whole `TableView`. Each `TableSection` can have its own `BindingContext`, which means that you can divide your `ViewModels` to coordinate more closely with the `TableView` layout.

A TableView menu

Besides displaying data or serving as a form or settings dialog, a `TableView` can also be a menu. Functionally, a menu is a collection of buttons, although they might not look like traditional buttons. Each menu item is a command that triggers a program operation.

This is why `TextCell` and `ImageCell` have `Command` and `CommandParameter` properties. These cells can trigger commands defined in a `ViewModel`, or simply some other property of type `ICommand`.

The XAML file in the `MenuCommands` program binds the `Command` properties of four `TextCell` elements with a property named `MoveCommand`, and passes to that `MoveCommand` arguments named "left", "up", "right", and "down":

```
<ContentPage xmlns = "http://xamarin.com/schemas/2014/forms"
             xmlns:x = "http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class = "MenuCommands.MenuCommandsPage"
             x:Name = "page">

    <ContentPage.Padding>
        <OnPlatform x:TypeArguments = "Thickness"
                   iOS = "0, 20, 0, 0" />
    </ContentPage.Padding>
```

```

< StackLayout >
    < TableView Intent = " Menu "
        VerticalOptions = " Fill "
        BindingContext = " {x:Reference page} " >
        < TableRoot >
            < TableSection Title = " Move the Box " >
                < TextCell Text = " Left "
                    Command = " {Binding MoveCommand} "
                    CommandParameter = " left " />

                < TextCell Text = " Up "
                    Command = " {Binding MoveCommand} "
                    CommandParameter = " up " />

                < TextCell Text = " Right "
                    Command = " {Binding MoveCommand} "
                    CommandParameter = " right " />

                < TextCell Text = " Down "
                    Command = " {Binding MoveCommand} "
                    CommandParameter = " down " />
            </ TableSection >
        </ TableRoot >
    </ TableView >

    < AbsoluteLayout BackgroundColor = " Maroon "
        VerticalOptions = " FillAndExpand " >
        < BoxView x:Name = " boxView "
            Color = " Blue "
            AbsoluteLayout.LayoutFlags = " All "
            AbsoluteLayout.LayoutBounds = " 0.5, 0.5, 0.2, 0.2 " />
    </ AbsoluteLayout >
</ StackLayout >
</ ContentPage >

```

But where is that `MoveCommand` property? If you look at the `BindingContext` of the `TableView`, you'll see that it references the root element of the XAML file, which means that `MoveCommand` property can probably be found as a property in the code-behind file.

And there it is:

```

public partial class MenuCommandsPage : ContentPage
{
    int xOffset = 0;           // ranges from -2 to 2
    int yOffset = 0;           // ranges from -2 to 2

    public MenuCommandsPage()
    {
        // Initialize ICommand property before parsing XAML.
        MoveCommand = new Command < string >(ExecuteMove, CanExecuteMove);

        InitializeComponent();
    }
}

```

```
public ICommand MoveCommand { private set; get; }

void ExecuteMove( string direction)
{
    switch (direction)
    {
        case "left" : xOffset--; break ;
        case "right" : xOffset++; break ;
        case "up" : yOffset--; break ;
        case "down" : yOffset++; break ;
    }

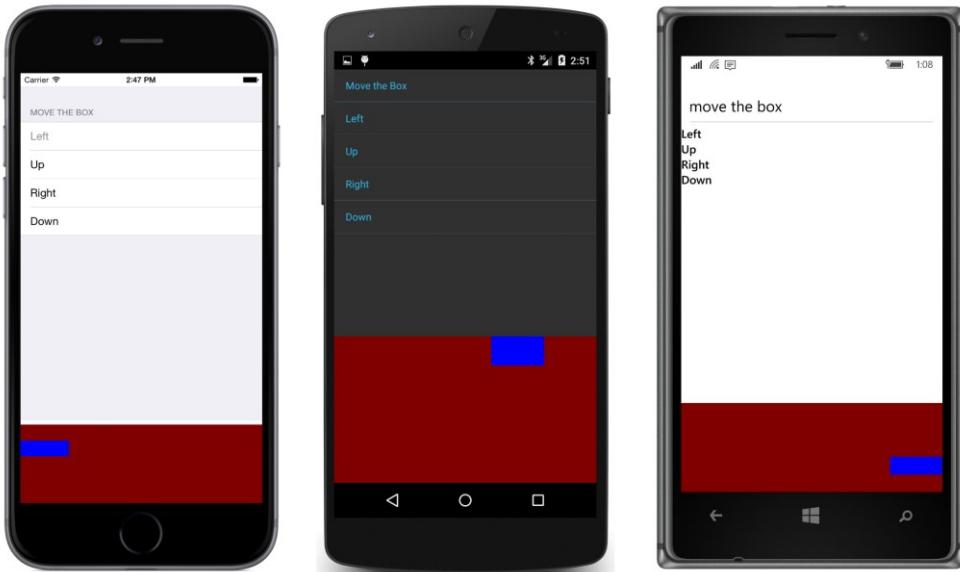
    (( Command )MoveCommand).ChangeCanExecute();

    AbsoluteLayout .SetLayoutBounds(boxView,
        new Rectangle ( (xOffset + 2) / 4.0,
                        (yOffset + 2) / 4.0, 0.2, 0.2));
}

bool CanExecuteMove( string direction)
{
    switch (direction)
    {
        case "left" : return xOffset > -2;
        case "right" : return xOffset < 2;
        case "up" : return yOffset > -2;
        case "down" : return yOffset < 2;
    }
    return false ;
}
}
```

The Execute method manipulates the layout bounds of a BoxView in the XAML file so that it moves around the AbsoluteLayout. The CanExecute method disables an operation if the BoxView has been moved to one of the edges.

Only on iOS does the disabled TextCell actually appear with a typical gray coloring, but on both the iOS and Android platforms the TextCell is no longer functional if the CanExecute method returns false:



You can also use `TableView` as a menu for page navigation or working with master/detail pages, and for these particular applications you might wonder whether a `ListView` or `TableView` is the right tool for the job. Generally it's `ListView` if you have a collection of items that should all be displayed in the same way, or `TableView` for fewer items that might require individual attention.

What is certain is that you'll definitely see more examples in the chapters ahead.

Chapter 20

Async and file I/O

Graphical user interfaces have a little peculiarity that has far-reaching consequences: User input to an application must be processed sequentially. Regardless of whether user-input events come from a keyboard, a mouse, or touch, each event must be completely processed by an application—either directly or through user-interface objects such as buttons or sliders—before the application obtains the next user-input event from the operating system.

The rationale behind this restriction becomes clear after a little reflection and perhaps an example: Suppose a page contains two buttons, and the user quickly taps one and then the other. Might it be possible for the two buttons to process those two taps concurrently in two separate threads of execution? No, that would not work. It could be that the first button changes the meaning of the second button, perhaps disabling it entirely. For this reason, the first button must be allowed to completely finish processing its tap before the second button begins processing its own tap.

The consequences of this restriction are severe: All user input to a particular application must be processed in a single thread of execution. Moreover, user-interface objects are generally not threadsafe. They cannot be modified from a secondary thread of execution. All code connected with an application's user interface is therefore restricted to a single thread. This thread is known as the *main thread* or the *user-interface thread* or the *UI thread*.

As we users have become more accustomed to graphical user interfaces over the decades, we've become increasingly intolerant of even the slightest lapse in responsiveness. As application programmers, we therefore try our best to keep the user interface responsive to achieve maximum user satisfaction. This means that anything running on the UI thread must perform its processing as quickly as possible and return control back to the operating system. If an event handler running in the UI thread gets bogged down in a long processing job, the entire user interface will seem to freeze and certainly annoy the user.

For this reason, any lengthy jobs that an application must perform should be relegated to secondary threads of execution, often called *worker threads*. These worker threads are said to run "in the background" and do not interfere with the responsiveness of the UI thread.

You've already seen some examples in this book. Several sample programs—the **ImageBrowser** and **BitmapStreams** programs in Chapter 13, "Bitmaps," and the **SchoolOfFineArt** library and **RssFeed** program in Chapter 19, "Collection views"—use the **WebRequest** class to download files over the Internet. A call to the **BeginGetResponse** method of **WebRequest** starts a worker thread that accesses the web resource asynchronously. The **WebRequest** call returns quickly, and the program can handle other user input while the file is being downloaded. An argument to **BeginGetResponse** is a callback method that is invoked when the background process completes. Within this callback method the program calls **EndGetResponse** to get access to the downloaded data.

But the callback method passed to `BeginGetResponse` has a little problem. The callback method runs in the same worker thread that downloads the file, and in the general case, you can't access userinterface objects from anything other than the UI thread. Usually, this means that the callback method must access the UI thread. Each of the three platforms supported by Xamarin.Forms has its own native method for running code from a secondary thread on the UI thread, but in Xamarin.Forms these are all available through the `Device.BeginInvokeOnMainThread` method. (As you'll recall, however, there are some exceptions generally related to ViewModels: Although a secondary thread can't access a user-interface object directly, the secondary thread can set a property that is bound to a user-interface object through a data binding.)

In recent years, asynchronous processing has become more ubiquitous at the same time that it's become easier for programmers. This is an ongoing trend: The future of computing will undoubtedly involve a lot more asynchronous computing and parallel processing, particularly with the increasing use of multicore processor chips. Developers will need good operating-system support and language tools to work with asynchronous operations, and fortunately .NET and C# have been in the forefront of this support.

This chapter will explore some of the basics of working with asynchronous processing in Xamarin.Forms applications, including using the `.NET Task` class to help you define and work with asynchronous methods. The customary hassle of dealing with callback functions has been alleviated greatly with two keywords introduced in C# 5.0: `async` and `await`. The `await` operator has revolutionized asynchronous programming by simplifying the syntax of asynchronous calls, by clarifying program flow surrounding asynchronous calls, by easing the access of user-interface objects, by simplifying the handling of exceptions raised by worker threads, and by unifying the handling of these exceptions and cancellations of background jobs.

This chapter primarily demonstrates how to work with asynchronous processing to perform file input and output, and how to create your own worker threads for performing lengthy jobs.

But Xamarin.Forms itself contains several asynchronous methods.

From callbacks to await

The `Page` class defines three methods that let you display a visual object sometimes called an *alert* or a *message box*. Such a box pops up on the screen with some information or a question for the user. The alert box is modal, meaning that the rest of the application is unavailable while the alert is displayed. The user must dismiss it with the press of a button before returning to interact with the application.

Two of these three methods of the `Page` class are named `DisplayAlert`. The first simply displays some text with a single button to dismiss the box, while the second contains two buttons for yes or no responses. The `DisplayActionSheet` method is similar but displays any number of buttons.

In iOS, Android, and the Windows Runtime, these methods are implemented with platform-specific

objects that use events or callback methods to inform the application that the alert box has been dismissed and what button the user pressed to dismiss it. However, Xamarin.Forms has wrapped these objects with an asynchronous interface.

These three methods of the **Page** class are defined like this:

```
Task DisplayAlert (string title, string message, string cancel)

Task<bool> DisplayAlert (string title, string message, string accept, string cancel)

Task<string> DisplayActionSheet (string title, string cancel, string destruction,
                                params string[] buttons)
```

They all return Task objects. The Task and Task<T> classes are defined in the System.Threading.Tasks namespace and they form the core of the Task-based Asynchronous Pattern, known as TAP. TAP is the recommended approach to handling asynchronous operations in .NET. The Task Parallel Library (TPL) builds on TAP.

In contrast, the **BeginGetResponse** and **EndGetResponse** methods of **WebRequest** represent an older approach to asynchronous operations involving **IAsyncResult**. This older approach is called the Asynchronous Programming Model or APM. You might also encounter code that uses the Event-based Asynchronous Model (EAP) to return information from asynchronous jobs through events.

You've already seen the simplest form of **DisplayAlert** in the **SetTimer** program in Chapter 15, "The interactive interface." **SetTimer** used an alert to indicate when a timer elapsed. The program didn't seem to care that **DisplayAlert** returned a Task object because the alert box was used strictly for notification purposes. It was not necessary to obtain a response from the user. However, the methods that return **Task<bool>** and **Task<string>** need to convey actual information back to the application indicating which button the user pressed to dismiss the alert.

A return value of **Task<T>** is sometimes referred to as a "promise." The actual value or object isn't available just yet, but it will be available in the future if nothing goes awry.

You can work with a **Task<T>** object in a few different ways. These approaches are fundamentally equivalent, but the C# syntax is quite different.

An alert with callbacks

The intended use of the **DisplayAlert** method that returns a **Task<bool>** is to ask the user a question with a yes or no answer. Obviously the answer isn't available until the user presses a button and the alert is dismissed, at which time a true value means Yes and false value means No.

One way to work with a **Task<T>** object is with callback methods. The **AlertCallbacks** program demonstrates that approach. It has a XAML file with a Button to invoke an alert and a Label for the program to display some information:

```
<ContentPage xmlns = "http://xamarin.com/schemas/2014/forms"
              xmlns:x = "http://schemas.microsoft.com/winfx/2009/xaml"
              x:Class = "AlertCallbacks.AlertCallbacksPage">
```

```

< StackLayout >
    < Button Text = " Invoke Alert "
        FontSize = " Large "
        HorizontalOptions = " Center "
        VerticalOptions = " CenterAndExpand "
        Clicked = " OnButtonClicked " />

    < Label x:Name = " label "
        Text = " Tap button to invoke alert "
        FontSize = " Large "
        HorizontalTextAlignment = " Center "
        VerticalOptions = " CenterAndExpand " />
</ StackLayout >
</ ContentPage >

```

Here's the code-behind file with the Clicked event handler and two callback methods:

```

public partial class AlertCallbacksPage : ContentPage
{
    bool result;

    public AlertCallbacksPage()
    {
        InitializeComponent();
    }

    void OnButtonClicked( object sender, EventArgs args )
    {
        Task< bool > task = DisplayAlert( "Simple Alert" , "Decide on an option" ,
                                         "yes or ok" , "no or cancel" );
        task.ContinueWith( AlertDismissedCallback );
        label.Text = "Alert is currently displayed" ;
    }

    void AlertDismissedCallback( Task< bool > task )
    {
        result = task.Result;
        Device.BeginInvokeOnMainThread( DisplayResultCallback );
    }

    void DisplayResultCallback()
    {
        label.Text = String.Format( "Alert {0} button was pressed" ,
                                   result ? "OK" : "Cancel" );
    }
}

```

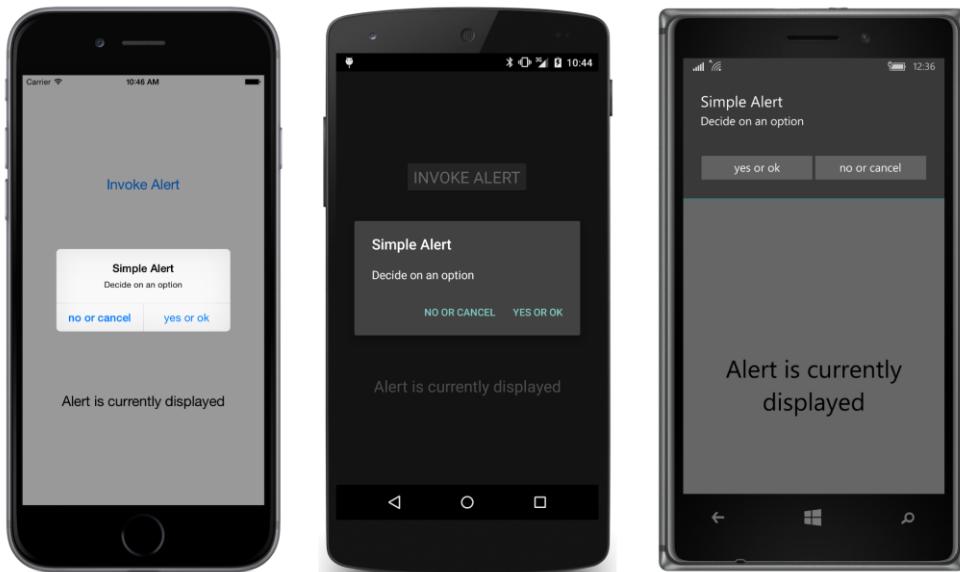
The Clicked handler calls `DisplayAlert` with arguments indicating a title, a question or statement, and the text for the two buttons. Generally, these two buttons are labeled “yes” and “no,” or “ok” and “cancel,” but you can put anything you want in those buttons as this program demonstrates.

If `DisplayAlert` were designed to be a synchronous method, the method would return a `bool` indicating which button the user pressed to dismiss the alert. However, `DisplayAlert` would not be

able to return that value until the alert were dismissed, which means that the application would be stuck in the `DisplayAlert` call during the entire time the alert is displayed. Depending on how the operating system handles user-input events, being stuck in the `DisplayAlert` call might not actually block other event handling by the user-interface thread during this time, but it might be a little strange for the UI thread to be seemingly in the `DisplayAlert` call while also handling other events.

Instead of returning a bool when the alert is dismissed, `DisplayAlert` returns a `Task<bool>` object that promises a bool result sometime in the future. To obtain that value, the `OnButtonClicked` handler in the **AlertCallbacks** program calls the `ContinueWith` method defined by `Task`. This method allows the program to specify a method that is called when the alert is dismissed. The `Clicked` handler concludes by setting some text to the Label, and then returns control back to the operating system.

The alert is then displayed:



Of course, the alert essentially disables the user interface of the application, but the application could still be doing some work while the alert is displayed. For example, the program could be using a timer, and that timer would continue to run. You can prove this to yourself by adding the following code to the constructor of the **AlertCallbacks** code-behind file:

```
Device.StartTimer(TimeSpan.FromSeconds(1), () =>
{
    label.Text = DateTime.Now.ToString();
    return true;
});
```

When the user dismisses the alert by tapping one of the buttons, the `AlertDismissedCallback` method is called:

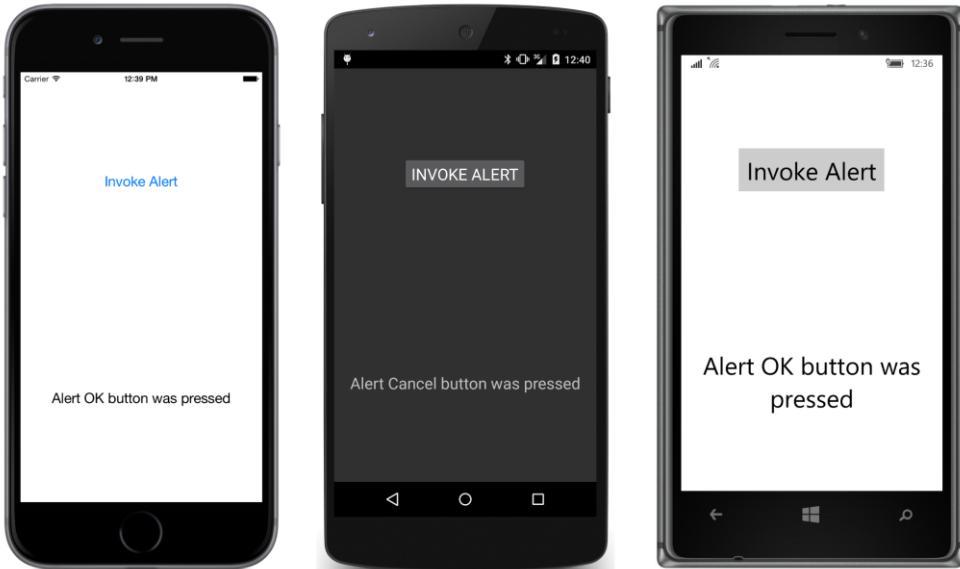
```
void AlertDismissedCallback( Task< bool > task)
{
    result = task.Result;
    Device.BeginInvokeOnMainThread(DisplayResultCallback);
}
```

The argument is the same Task object originally returned from the DisplayAlert method. But now the Result property of the Task object has been set to true or false depending on what button the user pressed to dismiss the alert. The program wants to display that value, but unfortunately it cannot because this AlertDismissedCallback method is running in a secondary thread that Xamarin.Forms has created. This thread is not allowed to access any user-interface objects of the program. For that reason, the AlertDismissedCallback method saves the bool result in a field and calls Device-

- . BeginInvokeOnMainThread with a second callback method. That callback method runs in the UI thread:

```
void DisplayResultCallback()
{
    label.Text = String.Format("Alert {(0)} button was pressed",
        result ? "OK" : "Cancel");
}
```

The Label then displays that text:



The **AlertCallbacks** program demonstrates one traditional way to handle asynchronous methods, but it has a distinct drawback: There are simply too many callbacks, and in one case, data must be passed from one callback to another by using a field.

An alert with lambdas

An obvious approach to simplify callbacks is with lambda functions. This is demonstrated with the **AlertLambdas** program. The XAML file is the same as in the **AlertCallbacks** method, but everything that happens in response to the button click is now inside that Clicked handler:

```
public partial class AlertLambdasPage : ContentPage
{
    public AlertLambdasPage()
    {
        InitializeComponent();
    }

    void OnButtonClicked( object sender, EventArgs args )
    {
        Task < bool > task = DisplayAlert( "Simple Alert", "Decide on an option",
                                         "yes or ok", "no or cancel" );
        task.ContinueWith( ( Task < bool > taskResult ) =>
        {
            Device.BeginInvokeOnMainThread( () =>
            {
                label.Text = String.Format( "Alert {0} button was pressed",
                                            taskResult.Result ? "OK" : "Cancel" );
            });
        });
        label.Text = "Alert is currently displayed";
    }
}
```

There is really no difference between this program and the previous one except that the callback methods have no name. They are anonymous. But sometimes lambda functions have the tendency to obscure program flow, and that is certainly the case here. The Text property of the Label is set to the text "Alert is currently displayed" right after the ContinueWith method is called and before the callback passed to ContinueWith executes, but that statement appears at the bottom of the method.

There should be a better way to denote what you want to happen without distorting program flow. That better way is called await.

An alert with await

The **AlertAwait** program has the same XAML file as **AlertCallbacks** and **AlertLambdas**, but the OnButtonClicked method is considerably simplified:

```
public partial class AlertAwaitPage : ContentPage
{
    public AlertAwaitPage()
    {
        InitializeComponent();
    }

    async void OnButtonClicked( object sender, EventArgs args )
    {
```

```
Task < bool > task = DisplayAlert("Simple Alert", "Decide on an option",  
                                    "yes or ok", "no or cancel");  
  
label.Text = "Alert is currently displayed";  
  
bool result = await task;  
  
label.Text = String.Format("Alert {0} button was pressed",  
                           result ? "OK" : "Cancel");  
  
}  
}
```

The key statement is this one:

```
bool result = await task;
```

That task variable is the Task<bool> object returned from DisplayAlert, but the await keyword seems to magically extract the Boolean result without any callbacks or lambdas.

The first thing you should know is that `await` doesn't actually wait for the alert to be dismissed! Instead, the C# compiler has performed a lot of surgery on the `OnButtonClicked` method. The method has basically been turned into a state machine. Part of the method is executed when the button is clicked, and part of the method is executed later. When the flow of execution hits the `await` keyword, the remainder of the `OnButtonClicked` method is skipped over for the moment. The `OnButton-`

Clicked method exits and returns control back to the operating system. From the perspective of the Button, the event handler has completed.

When the user dismisses the alert box, the remainder of the `OnButtonClicked` method resumes execution beginning with the assignment of the Boolean value to the `result` variable. In some circumstances, some optimizations can take place behind the scenes. For example, the flow of execution can just continue normally if the asynchronous operation completes immediately.

The await operator has another bonus: Notice that there's no use of `Device.BeginInvokeOnMainThread`. When the user dismisses the alert, the `OnButtonClicked` method automatically resumes execution in the user-interface thread, which means that it can access the `Label`. (In some cases, you might want to continue running in the background thread for performance reasons. If so, you can use the `ConfigureAwait` method of `Task` to do that. You'll see an example later in this chapter.)

The `await` keyword essentially converts asynchronous code into something that appears to be normal sequential imperative code. Of course, behind the scenes, there is really not much difference between this program and the two previous programs. In all three cases, the `OnButtonClicked` handler returns control back to the operating system when it displays the alert, and resumes execution when the alert is dismissed.

Simply for illustrative purposes, the three programs display some text immediately after the `DisplayAlert` method is called. If that isn't necessary, then the `DisplayAlert` call can be combined with the `await` operator to get rid of the explicit `Task<bool>` variable entirely:

This is how await commonly appears in code. DisplayAlert returns Task<bool> but the await operator effectively extracts the bool result after the background task has completed.

Indeed, you can use await much like you can any other operator, and it can appear inside a more complex expression. For example, if you don't need the statement that displays the text after the Dis- playAlert call, you can actually put both the await operator and DisplayAlert inside the final String.Format call:

```
async void OnButtonClicked( object sender, EventArgs args)
{
    label.Text = String .Format( "Alert {0} button was pressed" ,
        await DisplayAlert( "Simple Alert" , "Decide on an option" ,
            "yes or ok" , "no or cancel" ) ? "OK" : "Cancel" );
}
```

That might be a little difficult to read, but think of the combination of the await operator and the DisplayAlert method as a bool and the statement makes perfect sense.

You might have noticed that the OnButtonClicked method is marked with the `async` keyword. Any method in which you use `await` must be marked as `async`. However, the `async` keyword does not change the signature of the method. OnButtonClicked still qualifies as an event handler for the Clicked event.

But not every method can be an `async` method.

An alert with nothing

The simpler of the two DisplayAlert methods returns a `Task` object. It is intended to display some information to the user that doesn't require a response:

```
Task DisplayAlert (string title, string message, string cancel)
```

Generally, you'll want to use `await` with this simpler DisplayAlert method even though it doesn't return any information, and particularly if you need to perform some processing after it has been dismissed. The NothingAlert program has the same XAML file as the previous samples but displays this simpler alert box:

```
public partial class NothingAlertPage : ContentPage
{
    public NothingAlertPage()
    {
        InitializeComponent();
    }

    async void OnButtonClicked( object sender, EventArgs args )
    {
        label.Text = "Displaying alert box" ;
        await DisplayAlert( "Simple Alert" , "Click 'dismiss' to dismiss" , "dismiss" );
        label.Text = "Alert has been dismissed" ;
    }
}
```

Nothing appears to the left of the await operator because the return value of `DisplayAlert` is `Task` rather than `Task<T>` and no information is returned.

The first program in this book that used this simpler form of `DisplayAlert` was the `SetTimer` program in Chapter 15. Here's the timer callback method from that program (with the oddly named `@switch` variable so that it doesn't conflict with the `switch` keyword):

```
bool OnTimerTick()
{
    if (@switch.IsToggled && DateTime.Now >= triggerTime)
    {
        @switch.IsToggled = false;
        DisplayAlert("Timer Alert",
                    "The " + entry.Text + " timer has elapsed",
                    "OK");
    }
    return true;
}
```

The `DisplayAlert` call returns quickly, and the method continues to execute when the alert box is displayed. The `OnTimerTick` method then returns true, and a second later `OnTimerTick` is called again. Fortunately, the `Switch` is no longer toggled, so the program doesn't attempt to call `Display-`

`Alert` a second time. When the alert is dismissed, the user can again interact with the user interface, but no additional code is executed on its return.

What if you wanted to execute a little code after the alert box was dismissed? Try to put an `await` operator in front of `DisplayAlert` and identify the method with the `async` keyword:

```
// Will not compile!
async bool OnTimerTick()
{
    if (@switch.IsToggled && DateTime.Now >= triggerTime)
    {
        @switch.IsToggled = false;
        await DisplayAlert("Timer Alert",
                           "The " + entry.Text + " timer has elapsed",
                           "OK");
        // Some code to execute after the alert box is dismissed.
    }
    return true;
}
```

But as the comment says, this code will not compile.

Why not?

When the C# compiler encounters the `await` keyword, it constructs code so that the `OnTimerTick` callback returns to its caller. The remainder of the method then resumes execution when the alert box is dismissed. However, the `Device.StartTimer` method that invokes this callback is expecting the timer callback to return a Boolean value to determine whether it should call the callback again, and the C# compiler cannot construct code that returns a Boolean value because it doesn't know what that

Boolean value should be!

For this reason, methods that contain await operators are restricted to return types of void, Task, or Task<T>.

Event handlers usually have void return types. This is why the Clicked handler of a Button can contain await operators and be flagged with the `async` keyword. But the timer callback method returns a bool, and to use await within this method, the return value of the `OnTimerTick` method must be `Task<bool>`:

```
// Method compiles but Device.StartTimer does not!
async Task<bool> OnTimerTick()
{
    if (@switch.IsToggled && DateTime.Now >= triggerTime)
    {
        @switch.IsToggled = false;
        await DisplayAlert("Timer Alert",
                           "The " + entry.Text + " timer has elapsed",
                           "OK");
    }
    return true;
}
```

This method now contains entirely legal compilable code. When a method is defined to return `Task<T>`, the body of the method returns an object of type T and the compiler does the rest.

However, because the method now returns a `Task<bool>` object, code that calls this method must use `await` with the method (or call `ContinueWith` on the `Task` object) to obtain the Boolean value when the method completes execution. That's a problem for the `Device.StartTimer` call, which is not expecting the callback method to be asynchronous; it's expecting the callback method to return

`bool` rather than `Task<bool>`.

If you really did want to execute some code after the alert is dismissed in the `SetTimer` program, you should use `ContinueWith` for that code. The `await` operator is very useful, but it is not a panacea for every asynchronous programming problem.

The `await` operator can only be used in a method, and the method must have a return type of `void`, `Task`, or `Task<T>`. That's it. The get accessors of properties cannot use `await`, and they shouldn't be performing asynchronous operations anyway. Constructors cannot use `await` because constructors are not methods and have no return type. You cannot use `await` in the body of a lock

statement. C# 5 also prohibits using `await` in the catch or finally blocks of a try-catch-finally statement, but C# 6 lifts that restriction.

These restrictions turn out to be most severe for constructors. A constructor should complete promptly because nothing can really be done with an instance of a class until the constructor finishes. Although a constructor can call an asynchronous method that returns `Task`, the constructor can't use `await` with that call. The constructor finishes while the asynchronous method is still processing. (You'll see some examples in this chapter and the next.)

A constructor cannot call an asynchronous method that returns a value required by the constructor to complete. If a constructor needs to obtain an object from an asynchronous operation, it can use ContinueWith, in which case the constructor will finish before the object from the asynchronous operation is available. But that's unavoidable.

Saving program settings asynchronously

As you discovered in Chapter 6, "Button clicks," you can save program settings in a dictionary named **Properties maintained by the Application class. Anything you put in the Properties dictionary is saved when the program goes into a sleep state and is restored when the program resumes or starts up again. Sometimes it's convenient to save settings in this dictionary as they are changed, and sometimes it's convenient to wait until the OnSleep method is called in your App class.**

There's also another option: The Application class has a method named SavePropertiesAsync that lets your program take a more proactive role in saving program settings. This allows a program to save program settings whenever it wants to. If the program later crashes or is terminated through the Visual Studio or Xamarin Studio debugger, the settings are saved.

In conformance with recommended practice, the Async suffix on the SavePropertiesAsync method name identifies this as an asynchronous method. It returns quickly with a Task object and saves the settings in a secondary thread of execution.

A program named SaveProgramSettings demonstrates this technique. The XAML file contains four Switch views and four Label views that treat the Switch views as digits of a binary number:

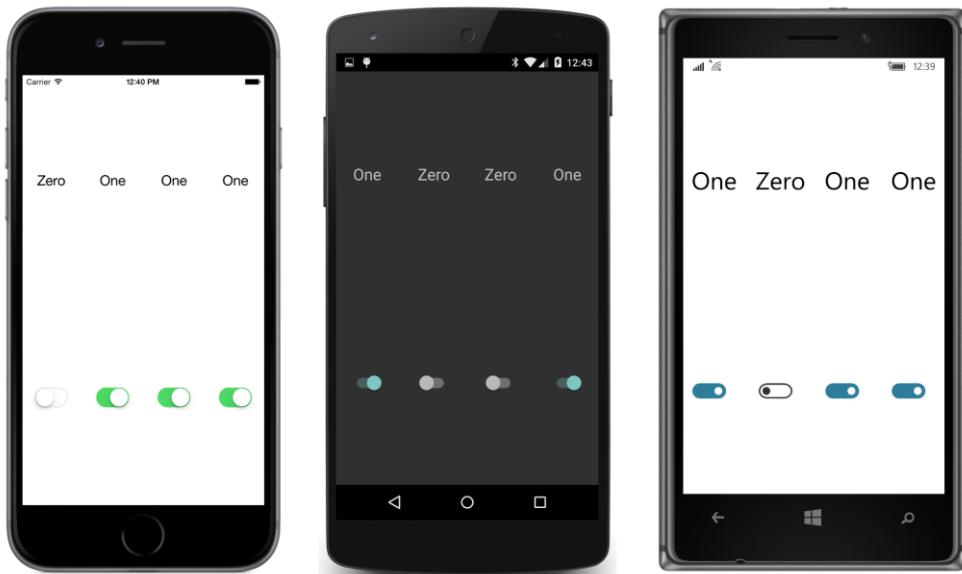
```
<ContentPage xmlns = "http://xamarin.com/schemas/2014/forms"
    xmlns:x = "http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit =
        "clr-namespace:Xamarin.FormsBook.Toolkit;assembly=Xamarin.FormsBook.Toolkit"
    x:Class = "SaveProgramSettings.SaveProgramSettingsPage">

    <ContentPage.Resources>
        <ResourceDictionary>
            <toolkit:BoolToStringConverter x:Key = "boolToString" 
                FalseText = "Zero"
                TrueText = "One" />
            <Style TargetType = "Label">
                <Setter Property = "FontSize" Value = "Large" />
                <Setter Property = "HorizontalTextAlignment" Value = "Center" />
            </Style>
            <Style TargetType = "Switch">
                <Setter Property = "HorizontalOptions" Value = "Center" />
            </Style>
        </ResourceDictionary>
    </ContentPage.Resources>

    <StackLayout>
        <Grid VerticalOptions = "CenterAndExpand" >
            <Label Text = "{Binding Source={x:Reference s3},
```

```
        Path=IsToggled,  
        Converter={StaticResource boolToString} "  
    Grid.Column = " 0 " />  
  
    < Label Text = " {Binding Source={x:Reference s2},  
        Path=IsToggled,  
        Converter={StaticResource boolToString} "  
    Grid.Column = " 1 " />  
  
    < Label Text = " {Binding Source={x:Reference s1},  
        Path=IsToggled,  
        Converter={StaticResource boolToString} "  
    Grid.Column = " 2 " />  
  
    < Label Text = " {Binding Source={x:Reference s0},  
        Path=IsToggled,  
        Converter={StaticResource boolToString} "  
    Grid.Column = " 3 " />  
  </ Grid >  
  
  < Grid x:Name = " switchGrid "  
    VerticalOptions = " CenterAndExpand " >  
    < Switch x:Name = " s3 " Grid.Column = " 0 "  
      Toggled = " OnSwitchToggled " />  
  
    < Switch x:Name = " s2 " Grid.Column = " 1 "  
      Toggled = " OnSwitchToggled " />  
  
    < Switch x:Name = " s1 " Grid.Column = " 2 "  
      Toggled = " OnSwitchToggled " />  
  
    < Switch x:Name = " s0 " Grid.Column = " 3 "  
      Toggled = " OnSwitchToggled " />  
  </ Grid >  
  </ StackLayout >  
</ ContentPage >
```

The data bindings on the Label elements allow them to track the values of the Switch views:



The saving and retrieving of program settings is handled in the code-behind file. Notice the handler assigned to the Toggled events of the Switch elements. The sole purpose of that handler is to store the settings in the Properties dictionary—and to save the Properties dictionary itself by using `SavePropertiesAsync`—whenever one of the Switch elements changes state. The dictionary key is the index of the Switch within the Children collection of the Grid:

```
public partial class SaveProgramSettingsPage : ContentPage
{
    bool isInitialized = false;

    public SaveProgramSettingsPage()
    {
        InitializeComponent();

        // Retrieve settings.
        IDictionary<string, object> properties = Application.Current.Properties;

        for (int index = 0; index < 4; index++)
        {
            Switch switcher = (Switch)(switchGrid.Children[index]);
            string key = index.ToString();

            if (properties.ContainsKey(key))
                switcher.IsToggled = (bool)(properties[key]);
        }
        isInitialized = true;
    }

    async void OnSwitchToggled(object sender, EventArgs args)
    {
        if (!isInitialized)
```

```

    return ;

    Switch switcher = ( Switch )sender;
    string key = switchGrid.Children.IndexOf(switcher).ToString();
    Application .Current.Properties[key] = switcher.IsToggled;

    // Save settings.
    foreach ( View view in switchGrid.Children)
        view.IsEnabled = false ;

    await Application .Current.SavePropertiesAsync();

    foreach ( View view in switchGrid.Children)
        view.IsEnabled = true ;
}

}
}

```

One of the purposes of this exercise is to emphasize first, that using await doesn't completely solve problems involved with asynchronous operations, but second, that using await can help deal with those potential problems.

Here's the problem: The Toggled event handler is called every time a Switch changes state. It could be that a user toggles a couple of the Switch views in succession very quickly. And it could also be the case that the SavePropertiesAsync method is slow. Perhaps it saves much more information than four Boolean values. Because this method is asynchronous, there is a danger that it could be called again while it's still working to save the previous collection of settings.

Is SavePropertiesAsync reentrant? Can it safely be called again while it's still working? We don't know, and it's better to assume that it's not. For that reason, the handler disables all the Switch elements before calling SavePropertiesAsync and then reenables them after it's finished. Because

SavePropertiesAsync returns Task rather than Task<T>, it's not necessary to use await (or ContinueWith) to get a value from the method, but it is necessary if you want to execute some code after the method has completed.

In reality, SavePropertiesAsync works so fast in this case that it's hard to tell whether this disabling and enabling of the Switch views is even working! For testing code such as this, a static method of the Task class is very useful. Try inserting this statement right after the SavePropertiesAsync call:

```
await Task .Delay(3000);
```

The Switch elements are disabled for another 3,000 milliseconds. Of course, if an asynchronous operation really took this long to complete and the user interface is disabled during this time, you'd want to display an ActivityIndicator or a ProgressBar if possible.

The Task.Delay method might seem reminiscent of the Thread.Sleep method that you possibly used in some .NET code many years ago. But the two static methods are very different. The Thread.Sleep method suspends the current thread, which in this case would be the user-interface thread. That's precisely what you *don't want*. The Task.Delay call, however, simulates a do-nothing secondary thread that runs for a specified period of time. The user-interface thread isn't blocked. If you

omit the await operator, Task.Delay would seemingly have no effect on the program at all. When used with the await operator, the code in the method that calls Task.Delay resumes after the specified period of time.

A platform-independent timer

So far in this book you've seen two ViewModels that have required timers: These are the `DateTime-ViewModel` class used in the `MvvmClock` program in Chapter 18, "MVVM," and the `SchoolViewModel` class in the `SchoolOfFineArt` library, which used the timer to randomly alter the students' grade-point averages for several programs in Chapter 19, "Collection views."

These ViewModels used `Device.StartTimer`, but that's not a good practice. A ViewModel is supposed to be platform independent and usable in any .NET application, but `Device.StartTimer` is specific to `Xamarin.Forms`.

You can alternatively create your own timer by using `Task.Delay`. Because `Task.Delay` is part of .NET and can be used within Portable Class Libraries, it is much more platform independent than `Device.StartTimer`.

The `TaskDelayClock` demonstrates how to use `Task.Delay` for a timer. The XAML file consists of a Label in an AbsoluteLayout:

```
<ContentPage xmlns = "http://xamarin.com/schemas/2014/forms"
    xmlns:x = "http://schemas.microsoft.com/winfx/2009/xaml"
    x:Class = "TaskDelayClock.TaskDelayClockPage">

    <ContentPage.Padding>
        <OnPlatform x:TypeArguments = "Thickness">
            iOS = "0, 20, 0, 0" />
    </ContentPage.Padding>

    <AbsoluteLayout>
        <Label x:Name = "label"
            FontSize = "Large"
            AbsoluteLayout.LayoutFlags = "PositionProportional" />
    </AbsoluteLayout>
</ContentPage>
```

The code-behind file contains a method called `InfiniteLoop`. Generally, infinite loops are avoided in programming, but this one runs in the user-interface thread for only a very brief period of time four times per second. For the bulk of the time, a `Task.Delay` call allows the user-interface thread to continue to interact with the user:

```
public partial class TaskDelayClockPage : ContentPage
{
    Random random = new Random();

    public TaskDelayClockPage()
    {
        InitializeComponent();
    }
```

```
InfiniteLoop();  
}  
  
async void InfiniteLoop()  
{  
    while (true)  
    {  
        label.Text = DateTime.Now.ToString("T");  
        label.FontSize = random.Next(12, 49);  
        AbsoluteLayout.SetLayoutBounds(label, new Rectangle (random.NextDouble(),  
            random.NextDouble(),  
            AbsoluteLayout.AutoSize,  
            AbsoluteLayout.AutoSize));  
        await Task.Delay(250);  
    }  
}
```

Every 250 milliseconds, the code in the while loop runs to give the Label the current time, but also to randomly change its font size and its location within the AbsoluteLayout:



Yes, it's a rather annoying clock.

This is not truly an “infinite” loop, of course, but it will keep going until the application terminates. If you prefer, you can use a Boolean field as the while conditional and exit from the loop by just setting the field to false.

Notice how the InfiniteLoop method is simply called from the constructor as if it were a normal method. If this method used Thread.Sleep rather than Task.Delay, it would never return back to

the constructor, and the constructor would never finish, and that would not be good at all. This particular `InfiniteLoop` method returns back to the constructor when execution hits the `await` operator for the first time, and the constructor can finish execution. The program can do anything else it wants, but the user-interface thread will be required every 250 milliseconds when `InfiniteLoop` resumes.

Although the `Task.Delay` call simulates a do-nothing secondary thread, it's actually implemented using the `Timer` class from the `System.Threading` namespace. Curiously enough, that `Timer` class is not available in a `Xamarin.Forms Portable Class Library`, and if it were, it would be a little more difficult to use because the timer callback doesn't run in the user-interface thread.

File input/output

Traditionally, file input/output is one of the most basic programming tasks, but file I/O on mobile devices is a little different from that on the desktop. On the desktop, users and applications generally have access to an entire disk and perhaps additional drives, all of which are organized into directory structures. On mobile devices, several standard folders exist—for pictures or music, for example—but application-specific data is generally restricted to a storage area that is private to each application.

Programmers familiar with .NET know that the `System.IO` namespace contains the bulk of standard file I/O support. This is where you'll find the crucial `Stream` class that provides the basis of reading and writing data organized as a stream of bytes.

Building upon this are several `Reader` and `Writer`

classes and other classes that allow accessing files and directories. Perhaps the handiest of the file classes is `File` itself, which not only provides a collection of methods to create new files and open existing files but also includes several static methods capable of performing an entire file-read or file-write operation in a single method call.

Particularly if you're working with text files, these static methods of the `File` class can be very convenient. For example, the `File.WriteAllText` method has two arguments of type `string` —a filename and the file contents. The method creates the file (replacing an existing file with the same name if necessary), writes the contents to the file, and then closes it. The `File.ReadAllText` method is similar but returns the contents of the file in one big `string` object. These methods are ideal for writing and reading text files with a minimum of fuss.

At first, file I/O doesn't seem to require asynchronous operations, and in practice, sometimes you have a choice, and sometimes you can avoid asynchronous operations if you want to.

However, other times you do not have a choice. Some platforms require asynchronous functions for file I/O, and even when they're not required, it makes sense to avoid doing file I/O in the user-interface thread.

Good news and bad news

The `Xamarin.iOS` and `Xamarin.Android` libraries referenced by your `Xamarin.Forms` applications include

a version of .NET that Xamarin has expressly tailored for these two mobile platforms. The methods in the `File` class in the `System.IO` namespace map to appropriate file I/O functions in the iOS and Android platforms, and the static `Environment.GetFolderPath` method, when used with the `MyDocu-`

ments enumeration member, returns a directory for the application's local storage. This means that you can use simple methods in the `File` class—including the static methods that perform entire file writing or reading operations in a single call—in your iOS and Android applications.

To verify the availability of these classes, let's experiment a little: Go into Visual Studio or Xamarin Studio and load any `Xamarin.Forms` solution created so far. Bring up one of the code files in the iOS or Android project. In a constructor or method, type the `System.IO` namespace name and then a period. You'll get a list of all the available types in the namespace. If you then type `File` and a period, you'll get all the static methods in the `File` class, including `WriteAllText` and `ReadAllText`.

In the Windows 8.1 and Windows Phone 8.1 projects, however, you're working with a version of .NET created by Microsoft specifically for these platforms. If you type `System.IO` and a period, you won't even see the `File` class at all! It doesn't exist! (However, you'll discover that it does exist in the UWP project.)

Now go into any code file in a `Xamarin.Forms` Portable Class Library project. As you'll recall, a PCL for `Xamarin.Forms` targets the following platforms:

- .NET Framework 4.5
- Windows 8
- Windows Phone 8.1
- Xamarin.Android
- Xamarin.iOS
- Xamarin.iOS (Classic)

As you might have already anticipated, the `System.IO` namespace in a PCL is also missing the `File` class. PCLs are configured to support multiple target platforms. Consequently, the APIs implemented within the PCL are necessarily an intersection of the APIs in these target platforms.

Beginning with Windows 8 and the Windows Runtime API, Microsoft completely revamped file I/O and created a whole new set of classes. Your Windows 8.1, Windows Phone 8.1, and UWP applications instead use classes in the `Windows.Storage` namespace for file I/O.

If you are targeting only iOS and Android in your `Xamarin.Forms` applications, you can share file I/O code between the two platforms. You can use the static `File` methods and everything else in `System.IO`.

If you also want to target one of the Windows or Windows Phone platforms, you'll want to make use of `DependencyService` (discussed in Chapter 9, "Platform-specific API calls") for different file I/O logic for each of the platforms.

A first shot at cross-platform file I/O

In the general case, you'll use DependencyService to give your Xamarin.Forms applications access to file I/O functions. As you know from the previous explorations into DependencyService, you can define the functions you want in an interface in the Portable Class Library project, while the code to implement these functions resides in separate classes in the individual platforms.

The file I/O functions developed in this chapter will be put to a good use in the **NoteTaker** application in Chapter 24, "Page navigation." For a first shot at file I/O, let's work with a much simpler solution, named **TextFileTryout**, that implements several functions to work with text files. Let's also restrict ourselves to getting this program running on iOS and Android and forget about the Windows platforms for the moment.

The first step in making use of DependencyService is creating an interface in the PCL that defines all the methods you'll need. Here is such an interface in the **TextFileTryout** project, named **IFileHelper**:

Helper:

```
namespace TextFileTryout
{
    public interface IFileHelper
    {
        bool Exists( string filename );

        void WriteText( string filename, string text );

        string ReadText( string filename );

        IEnumerable< string > GetFiles();

        void Delete( string filename );
    }
}
```

The interface defines functions to determine whether a file exists, to write and read entire text files in one shot, to enumerate all the files created by the application, and to delete a file. In each platform implementation, these functions are restricted to the private file area associated with the application.

You then implement this interface in each of the platforms. Here's the **FileHelper** class in the iOS project, complete with using directives and the required **Dependency** attribute:

```
using System;
using System.Collections.Generic;
using System.IO;
using Xamarin.Forms;

[ assembly : Dependency ( typeof (TextFileTryout.iOS. FileHelper ) )]

namespace TextFileTryout.iOS
{
    class FileHelper : IFileHelper
    {
```

```
public bool Exists( string filename)
{
    string filepath = GetFilePath(filename);
    return File.Exists(filepath);
}

public void WriteText( string filename, string text)
{
    string filepath = GetFilePath(filename);
    File.WriteAllText(filepath, text);
}

public string ReadText( string filename)
{
    string filepath = GetFilePath(filename);
    return File.ReadAllText(filepath);
}

public IEnumerable< string > GetFiles()
{
    return Directory.GetFiles(GetDocsPath());
}

public void Delete( string filename)
{
    File.Delete(GetFilePath(filename));
}

// Private methods.

string GetFilePath( string filename)
{
    return Path.Combine(GetDocsPath(), filename);
}

string GetDocsPath()
{
    return Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);
}
}
```

It is essential that this class explicitly implements the `IFileHelper` interface and includes a Dependency attribute with the name of the class. These allow the `DependencyService` class in `Xamarin.Forms` to find this implementation of `IFileHelper` in the platform project. Two private methods at the bottom allow the program to construct a fully qualified filename using the directory of the application's private storage available from the `Environment.GetFolderPath` method.

In both `Xamarin.iOS` and `Xamarin.Android`, the implementation of `Environment.GetFolderPath` obtains the platform-specific area of the application's local storage, although the directory names that the method returns for the two platforms are very different.

As a result, the `FileHelper` class in the Android project is exactly the same as the one in the iOS

project apart from the different namespace names.

The iOS and Android versions of FileHelper make use of the static shortcut methods in the File class and a simple static method of Directory for obtaining all the files stored with the application. However, the implementation of IFileHelper in the Windows 8.1 and Windows Phone 8.1 projects can't use the shortcut methods in the File class because they are not available, and the Environment.GetFolderPath method isn't available in the UWP project.

Moreover, applications written for these Windows platforms should instead use file I/O functions implemented in the Windows Runtime API. Because the file I/O functions in the Windows Runtime are asynchronous, they do not fit into the interface established by the IFileHelper interface. For that reason, the version of FileHelper in the three Windows projects is forced to leave the crucial methods unimplemented. Here's the version in the UWP project:

```
using System;
using System.Collections.Generic;
using Xamarin.Forms;

[assembly : Dependency ( typeof (TextFileTryout.UWP. FileHelper ) )]

namespace TextFileTryout.UWP
{
    class FileHelper : IFileHelper
    {
        public bool Exists( string filename )
        {
            return false ;
        }

        public void WriteText( string filename, string text )
        {
            throw new NotImplementedException ( "Writing files is not implemented" );
        }

        public string ReadText( string filename )
        {
            throw new NotImplementedException ( "Reading files is not implemented" );
        }

        public IEnumerable < string > GetFiles()
        {
            return new string [0];
        }

        public void Delete( string filename )
        {
        }
    }
}
```

The version of FileHelper in the Windows 8.1 and Windows Phone 8.1 projects is identical except for the namespace name.

Normally, an application needs to reference the methods in each platform by using the `DependencyService.Get` method. However, the **TextFileTryout** program has made things easy for itself by defining a class named `FileHelper` in the PCL project that also implements `IFileHelper`, but incorporates the call to the `Get` method of `DependencyService` to call the platform versions of these methods:

```
namespace TextFileTryout
{
    class FileHelper : IFileHelper
    {
        IFileHelper fileHelper = DependencyService.Get< IFileHelper >();

        public bool Exists( string filename )
        {
            return fileHelper.Exists(filename);
        }

        public void WriteText( string filename, string text )
        {
            fileHelper.WriteText(filename, text);
        }

        public string ReadText( string filename )
        {
            return fileHelper.ReadText(filename);
        }

        public IEnumerable< string > GetFiles()
        {
            IEnumerable< string > filepaths = fileHelper.GetFiles();
            List< string > filenames = new List< string >();

            foreach ( string filepath in filepaths )
            {
                filenames.Add( Path.GetFileName(filepath));
            }
            return filenames;
        }

        public void Delete( string filename )
        {
            fileHelper.Delete(filename);
        }
    }
}
```

Notice that the `GetFiles` method performs a little surgery on the filenames returned from the platform implementation. The filenames that are obtained from the platform implementations of `GetFiles` are fully qualified, and while it might be interesting to see the folder names that iOS and Android use for application local storage, those filenames are going to be displayed in a `ListView` where the folder names will just be a distraction, so this `GetFiles` method strips off the file path.

The TextFileTryoutPage class tests these functions. The XAML file includes an Entry for a filename, an Editor for the file contents, a Button labeled "Save", and a ListView with all the previously saved filenames:

```
< ContentPage xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns:x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x:Class = " TextFileTryout.TextFileTryoutPage " >

    < ContentPage.Padding >
        < OnPlatform x:TypeArguments = " Thickness "
            iOS = " 0, 20, 0, 0 " />
    </ ContentPage.Padding >

    < Grid >
        < Grid.RowDefinitions >
            < RowDefinition Height = " Auto " />
            < RowDefinition Height = " * " />
            < RowDefinition Height = " Auto " />
            < RowDefinition Height = " * " />
        </ Grid.RowDefinitions >

        < Entry x:Name = " filenameEntry "
            Grid.Row = " 0 "
            Placeholder = " filename " />

        < Editor x:Name = " fileEditor "
            Grid.Row = " 1 " >
            < Editor.BackgroundColor >
                < OnPlatform x:TypeArguments = " Color "
                    WinPhone = "#D0D0D0 " />
            </ Editor.BackgroundColor >
        </ Editor >

        < Button x:Name = " saveButton "
            Text = " Save "
            Grid.Row = " 2 "
            HorizontalOptions = " Center "
            Clicked = " OnSaveButtonClicked " />

        < ListView x:Name = " fileListView "
            Grid.Row = " 3 "
            ItemSelected = " OnFileListViewItemSelected " >
            < ListView.ItemTemplate >
                < DataTemplate >
                    < TextCell Text = "(Binding)" >
                        < TextCell.ContextActions >
                            < MenuItem Text = " Delete "
                                IsDestructive = " True "
                                Clicked = " OnDeleteMenuItemClicked " />
                        </ TextCell.ContextActions >
                    </ TextCell >
                </ DataTemplate >
            </ ListView.ItemTemplate >
        </ ListView >
    </ Grid >
```

```
</ContentPage>
```

Just to keep things simple, all processing is performed in the code-behind file without a ViewModel. The code-behind file implements all the event handlers from the XAML file. The **Save** button checks whether the file exists first and displays an alert box if it does. Selecting one of the files in the **ListView** loads it in. In addition, the **ListView** implements a context menu to delete a file. All the file I/O functions are methods of the **FileHelper** class defined in the PCL and instantiated as a field at the top of the class:

```
public partial class TextFileTryoutPage : ContentPage
{
    FileHelper fileHelper = new FileHelper();

    public TextFileTryoutPage()
    {
        InitializeComponent();

        RefreshListView();
    }

    async void OnSaveButtonClicked( object sender, EventArgs args )
    {
        string filename = filenameEntry.Text;

        if (fileHelper.Exists(filename))
        {
            bool okResponse = await DisplayAlert( "TextFileTryout",
                "File " + filename +
                " already exists. Replace it?",
                "Yes" , "No" );

            if (okResponse)
                return ;
        }

        string errorMessage = null ;

        try
        {
            fileHelper.WriteText(filenameEntry.Text, fileEditor.Text);
        }
        catch ( Exception exc )
        {
            errorMessage = exc.Message;
        }

        if (errorMessage == null )
        {
            filenameEntry.Text = "";
            fileEditor.Text = "";
            RefreshListView();
        }
        else
        {
```

```

        await DisplayAlert( "TextFileTryout" , errorMessage, "OK" );
    }

}

async void OnFileListViewItemSelected( object sender, SelectedItemChangedEventArgs args )
{
    if (args.SelectedItem == null )
        return ;

    string filename = ( string )args.SelectedItem;
    string errorMessage = null ;

    try
    {
        fileEditor.Text = fileHelper.ReadText(( string )args.SelectedItem);
        filenameEntry.Text = filename;
    }
    catch ( Exception exc )
    {
        errorMessage = exc.Message;
    }

    if (errorMessage != null )
    {
        await DisplayAlert( "TextFileTryout" , errorMessage, "OK" );
    }
}

void onDeleteMenuItemClicked( object sender, EventArgs args )
{
    string filename = ( string )( MenuItem )sender.BindingContext;
    fileHelper.Delete(filename);
    RefreshListView();
}

void RefreshListView()
{
    fileListView.ItemsSource = fileHelper.GetFiles();
    fileListView.SelectedItem = null ;
}
}

```

The code-behind file calls `DisplayAlert` with the `await` operator on three occasions: The **Save** button uses `DisplayAlert` if the filename you specify already exists. This confirms that your real intention is to replace an existing file. The other two uses are for notification purposes for errors that occur when files are saved or loaded. The file save and file load operations are in `try` and `catch` blocks to catch any errors that might occur. The file save operation will fail for an illegal filename, for example. It is less likely that an error will be encountered on reading a file, but the program checks anyway.

The alerts that notify the user of an error could conceivably be displayed without the `await` operator, but they use `await` anyway to demonstrate a basic principle involved in exception handling: Although C# 6 allows using `await` in a `catch` block, C# 5 does not. To get around this restriction, the

catch block simply saves the error message in a variable called errorMessage, and then the code following the catch block uses DisplayAlert to display that text if it exists. This structure allows these event handlers to conclude with different processing depending on successful completion or an error.

Notice also that the constructor concludes with a call to RefreshListView to display all the existing files in the ListView, and the code-behind file also calls that method when a new file has been saved or a file has been deleted.

However, this program does not work on the Windows platforms. Let's fix that.

Accommodating Windows Runtime file I/O

The Windows Runtime API defined a whole new array of file I/O classes. Part of the impetus for this was the recognition of an industry-wide transition away from the relatively unconstrained file access of desktop applications toward a more sandboxed environment.

Much of the new file I/O API can be found in the Windows Runtime namespaces Windows.Storage and Windows.Storage.Streams. To store data that is private to an application, a Windows Runtime program first gets a special StorageFolder object:

```
StorageFolder localFolder = ApplicationData.Current.LocalFolder;
```

ApplicationData defines a static property named Current that returns the ApplicationData object for the application. LocalFolder is an instance property of ApplicationData.

StorageFolder defines methods named CreateFileAsync to create a new file and GetFileAsync to open an existing file. These two methods obtain objects of type StorageFile. With that object, a program can open the file for writing or reading with OpenAsync or OpenReadAsync. These methods obtain an IRandomAccessStream object. From this, DataWriter or DataReader objects are created to perform write or read operations.

This sounds a bit lengthy, and it is. Rather simpler approaches involve static methods of the FileIO class, which are similar to the static methods of the .NET File class. For text files, for example, FileIO.ReadTextAsync and FileIO.WriteTextAsync open a file, perform the read or write access, and close the file in one shot. The first argument to these methods is a StorageFile object.

At any rate, by this time you've undoubtedly noticed the frequent Async suffixes on these method names. Internally, all these methods spin off secondary threads of execution for doing the actual work and return quickly to the caller. The work takes place in the background, and the caller is notified of completion (or error) through callback functions.

Why is this?

When Windows 8 was first being created, the Microsoft developers took a good, hard look at timing and decided that any function call that requires more than 50 milliseconds to execute should be made asynchronous so that it would not interfere with the responsiveness of the user interface. APIs that require more than 50 milliseconds obviously include the file I/O functions, which often need to access

potentially slow pieces of hardware like disk drives or a network. Any Windows Runtime file I/O method that could possibly cause a physical storage device to be accessed was made asynchronous and given an **Async** suffix.

However, these asynchronous methods do *not* return Task objects. In the Windows Runtime, methods that return data have return types of `IAsyncOperation<TResult>`, while methods that do not return information have return types of `IAsyncAction`. These interfaces can all be found in the `System.Foundations` namespace.

Although these interfaces are not the same as `Task` and `Task<T>`, they are similar, and you can use `await` with them. You can also convert between the two asynchronous protocols. The `System.Runtime.WindowsRuntime` assembly includes a `System` namespace with a `WindowsRuntimeSystemExtensions` class that has extension methods named `AsAsyncAction`, `AsAsyncOpertion`, and `AsTask` that perform these conversions.

Let's rework the `TextFileTryout` program to accommodate asynchronous file I/O. The revised program is called `TextFileAsync` and is developed in the next section. Because asynchronous file I/O functions in the Windows projects will be accessed, all the file functions in the `IFileHelper` interface are defined to return `Task` or `Task<T>` objects.

Platform-specific libraries

Every programmer knows that potentially reusable code should be put in a library, and this is also the case for code used with dependency services. The asynchronous file I/O functions developed here will be reused in the `NoteTaker` program in Chapter 24, and you might want to use these functions in your own applications or perhaps develop your own functions.

However, these file I/O classes can't be put in just one library. Each of the various platform implementations of `FileHelper` must be in a library for that specific platform. This requires separate libraries for each platform.

The **Libraries** directory of the downloadable code for this book contains a solution named `Xamarin.FormsBook.Platform`. The **Platform** part of the name was inspired by the various `Xamarin-.Forms.Platform` libraries. Each of the various platforms is a separate library in this solution.

The `Xamarin.FormsBook.Platform` solution contains no fewer than seven library projects, each of which was created somewhat differently:

- `Xamarin.FormsBook.Platform` is a normal `Xamarin.Forms` Portable Class Library with a profile of 111, which means that it can be accessed by all the platforms. You can create such a library in Visual Studio by selecting **Cross Platform** at the left of the **Add New Project** dialog, and **Class Library (Xamarin.Forms)** in the central area. In the `Xamarin Studio New Project` dialog, select **Multiplatform** and **Library** at the left, and `Xamarin.Forms` and `Class Library` in the central area.

- **Xamarin.FormsBook.Platform.iOS** was created in Visual Studio by selecting **iOS** in the left column of the **Add New Project** dialog, and **Class Library (iOS)** in the central section. In Xamarin Studio select **iOS** and **Library** in the **New Project** dialog, and **Class Library** in the central area.
- **Xamarin.FormsBook.Platform.Android** was created in Visual Studio by selecting **Android** at the left of the **Add New Project** dialog and **Class Library (Android)** in the central section. In Xamarin Studio, select **Android** and **Library** at the left and **Class Library** in the central section.
- **Xamarin.FormsBook.Platform.UWP** is a library for Windows 10 and Windows 10 Mobile. It was created in Visual Studio by selecting **Windows** and **Universal** at the left, and then **Class Library (Universal Windows)**.
- **Xamarin.FormsBook.Platform.Windows** is a Portable Class Library just for Windows 8.1. It was created in Visual Studio by selecting **Windows**, **Windows 8**, and **Windows** at the left, and then **Class Library (Windows 8.1)**.
- **Xamarin.FormsBook.Platform.WinPhone** is a Portable Class Library just for Windows Phone 8.1. It was created in Visual Studio by selecting **Windows**, **Windows 8**, and **Windows Phone** at the left, and then **Class Library (Windows Phone)**.
- You'll often find that the three Windows platforms can share code because they all use variants of the Windows Runtime API. For this reason, a seventh project was created named **Xamarin.FormsBook.Platform.WinRT**. This is a shared project, and it was created in Visual Studio by searching for "Shared" in the **Add New Project** dialog, and selecting the **Shared Project** for C#.

If you're creating such a solution yourself, you'll also need to use the **Manage Packages for Solution** dialog to install the appropriate Xamarin.Forms NuGet packages for all these libraries.

You'll also need to establish references between the various projects in the solution. All the individual platform projects (with the exception of **Xamarin.FormsBook.Platform.WinRT**) need a reference to **Xamarin.FormsBook.Platform**. You set these references in the **Reference Manager** dialog by selecting **Solution** at the left. In addition, the three Windows projects (**UWP**, **Windows**, and **WinPhone**)

all need references to the shared **Xamarin.FormsBook.Platform.WinRT** project. You set these references in the **Reference Manager** dialog by selecting **Shared Projects** at the left.

All the projects have a static `Toolkit.Init` method. Here's the one in the **Xamarin.FormsBook.Platform** library:

```
namespace Xamarin.FormsBook.Platform
{
    public static class Toolkit
    {
        public static void Init()
        {
        }
    }
}
```

```
}
```

Most of the others are similar except that the version in the Android library actually saves some information that might be useful to classes implemented in this library:

```
namespace Xamarin.FormsBook.Platform.Android
{
    public static class Toolkit
    {
        public static void Init(Activity activity, Bundle bundle)
        {
            Activity = activity;
        }

        public static Activity Activity { private set; get; }
    }
}
```

The `Toolkit.Init` method in each of the Windows platforms calls a do-nothing `Toolkit.Init` method in the shared `Xamarin.FormsBook.Platform.WinRT` project:

```
namespace Xamarin.FormsBook.Platform.UWP
{
    public static class Toolkit
    {
        public static void Init()
        {
            Xamarin.FormsBook.Platform.WinRT.Toolkit.Init();
        }
    }
}
```

The purpose of these methods is to ensure that the libraries are bound to the application even if the application does not directly access anything in the library. It is very often the case when you're working with dependency services and custom renderers that the application does not directly call any library function. However, if you later discover that you really do need to perform some library initialization, the method already exists for you to do so.

You'll discover that the version of the `Xamarin.FormsBook.Platform` libraries included with the downloadable code for this book already includes the `PlatformSoundPlayer` classes from Chapter 9, "Platform-specific API calls." You'll also see some classes beginning with the words `Ellipse` and `StepSlider`. These are discussed in Chapter 27, "Custom renderers."

Let's focus on the new asynchronous `FileHelper` classes. The `Xamarin.FormsBook.Platform` library contains the new `IFileHelper` interface:

```
using System.Collections.Generic;
using System.Threading.Tasks;

namespace Xamarin.FormsBook.Platform
{
    public interface IFileHelper
```

```
{  
    Task < bool > ExistsAsync( string filename);  
  
    Task WriteTextAsync( string filename, string text);  
  
    Task < string > ReadTextAsync( string filename);  
  
    Task < IEnumerable < string >> GetFilesAsync();  
  
    Task DeleteAsync( string filename);  
}  
}
```

By convention, methods that return Task objects have a suffix of Async.

All three Windows platforms can share the same FileHelper class, so this shared class is implemented in the shared **Xamarin.FormsBook.Platform** project. Each of the five methods in the **FileHelper** class begins with a call to obtain the StorageFolder associated with the application's local storage area. Each of them makes asynchronous calls using await and is flagged with the **async** keyword:

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Threading.Tasks;  
using Windows.Storage;  
using Xamarin.Forms;  
  
[assembly : Dependency ( typeof (Xamarin.FormsBook.Platform.WinRT. FileHelper ))]  
  
namespace Xamarin.FormsBook.Platform.WinRT  
{  
    class FileHelper : IFileHelper  
    {  
        public async Task < bool > ExistsAsync( string filename)  
        {  
            StorageFolder localFolder = ApplicationData .Current.LocalFolder;  
  
            try  
            {  
                await localFolder.GetFileAsync(filename);  
            }  
            catch  
            {  
                return false ;  
            }  
            return true ;  
        }  
  
        public async Task WriteTextAsync( string filename, string text)  
        {  
            StorageFolder localFolder = ApplicationData .Current.LocalFolder;  
            IStorageFile storageFile = await localFolder.CreateFileAsync(filename,
```

```
CreationCollisionOption .ReplaceExisting);

        await FileIO .WriteTextAsync(storageFile, text);
    }

    public async Task < string > ReadTextAsync( string filename)
    {
        StorageFolder localFolder = ApplicationData .Current.LocalFolder;
        IStorageFile storageFile = await localFolder.GetFileAsync(filename);
        return await FileIO .ReadTextAsync(storageFile);
    }

    public async Task < IEnumerable < string >> GetFilesAsync()
    {
        StorageFolder localFolder = ApplicationData .Current.LocalFolder;
        IEnumerable < string > filenames =
            from storageFile in await localFolder.GetFilesAsync()
            select storageFile.Name;

        return filenames;
    }

    public async Task DeleteAsync( string filename)
    {
        StorageFolder localFolder = ApplicationData .Current.LocalFolder;
        StorageFile storageFile = await localFolder.GetFileAsync(filename);
        await storageFile.DeleteAsync();
    }
}
```

Although each of the methods is defined as returning a Task or a Task<T> object, the bodies of the methods don't have any reference to Task or Task<T>. Instead, the methods that return a Task object simply do some work and then end the method with an implicit return statement. The ExistsAsync method is defined as returning a Task<bool> but returns either true or false. (There is no Exists method in the StorageFolder class, so a workaround with try and catch is necessary.)

Similarly, the `ReadTextAsync` method is defined as returning a `Task<string>`, but the body returns a string, which is obtained from applying the `await` operator to the `IAsyncOperation<string>` return value of `File.ReadTextAsync`. The C# compiler performs the necessary conversions.

When a program calls this `ReadTextAsync` method, the method executes until the first `await` operator, and then it returns a `Task<string>` object to the caller. The caller can use either `ContinueWith` or `await` to obtain the string when the `FileIO.ReadTextAsync` method has completed.

For iOS and Android, however, we now have a problem. All the methods in `IFileHelper` are now defined as asynchronous methods that return `Task` or `Task<T>` objects, but we've already seen that the methods in the `System.IO` namespace are not asynchronous. What do we do?

The `FileHelper` class in the `iOS` namespace uses two strategies. In some cases, the `System.IO`

classes *do* include asynchronous methods. This is the case for the `WriteAsync` method of `StreamWriter` and the `ReadAsync` method of `StreamReader`. For the other methods, however, a static `FromResult` method of `Task<T>` is used to convert an object or value to a `Task<T>` object for the method return value. This does not actually convert the method to an asynchronous method, but simply allows the method to have the signature of an asynchronous method:

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Threading.Tasks;
using Xamarin.Forms;

[assembly : Dependency ( typeof (Xamarin.FormsBook.Platform.iOS. FileHelper ))]

namespace Xamarin.FormsBook.Platform.iOS
{
    class FileHelper : IFileHelper
    {
        public Task < bool > ExistsAsync( string filename )
        {
            string filepath = GetFilePath(filename);
            bool exists = File .Exists(filepath);
            return Task < bool > .FromResult(exists);
        }

        public async Task WriteTextAsync( string filename, string text )
        {
            string filepath = GetFilePath(filename);
            using ( StreamWriter writer = File .CreateText(filepath) )
            {
                await writer.WriteAsync(text);
            }
        }

        public async Task < string > ReadTextAsync( string filename )
        {
            string filepath = GetFilePath(filename);
            using ( StreamReader reader = File .OpenText(filepath) )
            {
                return await reader.ReadToEndAsync();
            }
        }

        public Task < IEnumerable < string >> GetFilesAsync()
        {
            // Sort the filenames.
            IEnumerable < string > filenames =
                from filepath in Directory .EnumerateFiles(GetDocsFolder())
                select Path .GetFileName(filepath);

            return Task < IEnumerable < string >> .FromResult(filenames);
        }
    }
}
```

```
public Task DeleteAsync( string filename )
{
    File .Delete(GetFilePath(filename));
    return Task .FromResult( true );
}

string GetDocsFolder()
{
    return Environment .GetFolderPath( Environment .SpecialFolder .MyDocuments);
}

string GetFilePath( string filename )
{
    return Path .Combine(GetDocsFolder(), filename);
}
}
```

The Android FileHelper class is the same as the iOS class but with a different namespace.

Notice that the only error checking within these platform implementations is for the ExistsAsync method in the Windows Runtime platforms, which uses the exception to determine whether the file exists or not. None of the other methods—and particularly the WriteTextAsync and ReadTextAsync methods—is performing any error checking. One of the nice features of using await is that any exception can be caught at a later time when you’re actually calling these methods.

You might also have noticed that the individual GetFilesAsync methods are now removing the path from the fully qualified filename, so that job doesn’t need to be performed by the FileHelper class in the **Xamarin.FormsBook.Platform** project:

```
namespace Xamarin.FormsBook.Platform
{
    class FileHelper
    {
        IFileHelper fileHelper = DependencyService .Get< IFileHelper >();

        public Task < bool > ExistsAsync( string filename )
        {
            return fileHelper.ExistsAsync(filename);
        }

        public Task WriteTextAsync( string filename, string text )
        {
            return fileHelper.WriteTextAsync(filename, text);
        }

        public Task < string > ReadTextAsync( string filename )
        {
            return fileHelper.ReadTextAsync(filename);
        }

        public Task < IEnumerable < string >> GetFilesAsync()
        {
        }
    }
}
```

```
        {
            return fileHelper.GetFilesAsync();
        }

        public Task DeleteAsync(string filename)
        {
            return fileHelper.DeleteAsync(filename);
        }
    }
}
```

Now that we have a library, we need to access this library from an application. The **TextFileAsync** solution was created normally. Then, all seven projects in the **Xamarin.FormsBook.Platform** solution were added to this solution. These projects must be added separately by using the **Add and Existing Project** menu item for the solution. There is no **Add All Projects from Solution** menu item, but if you use these libraries in your own projects, you'll wish there were!

At this point, the **TextFileAsync** solution contains 13 projects: Five application projects, a shared PCL with the application code, and seven library projects.

References must be established between these projects by using the **Reference Manager** for the following relationships:

- **TextFileAsync** has a reference to **Xamarin.FormsBook.Platform**.
- **TextFileAsync.iOS** has a reference to **Xamarin.FormsBook.Platform.iOS**.
- **TextFileAsync.Droid** has a reference to **Xamarin.FormsBook.Platform.Android**.
- **TextFileAsync.UWP** has a reference to **Xamarin.FormsBook.Platform.UWP**.
- **TextFileAsync.Windows** has a reference to **Xamarin.FormsBook.Platform.Windows**.
- **TextFileAsync.WinPhone** has a reference to **Xamarin.FormsBook.Platform.WinPhone**.

Of course, all the application projects have normal references to the **TextFileAsync** PCL, and, as you'll recall, the **Xamarin.FormsBook.Platform**, **Windows**, and **WinPhone** projects all have references to the shared **Xamarin.FormsBook.Platform.WinRT** project.

Also, all the **TextFileAsync** projects should make calls to the various **Toolkit.Init** methods in the libraries. In the **TextFileAsync** project itself, make the call in the constructor of the **App** class:

```
namespace TextFileAsync
{
    public class App : Application
    {
        public App()
        {
            Xamarin.FormsBook.Platform.Toolkit.Init();
            ...
        }
        ...
    }
}
```

```

        }
    }
}
```

In the iOS project, make the call after the normal Forms.Init call in the AppDelegate class:

```

namespace TextFileAsync.iOS
{
    ...
    public partial class AppDelegate : global::Xamarin.Forms.Platform.iOS.FormsApplicationDelegate
    {
        ...
        public override bool FinishedLaunching(UIApplication app, NSDictionary options)
        {
            global::Xamarin.Forms.Forms.Init();
            Xamarin.FormsBook.Platform.iOS.Toolkit.Init();
            LoadApplication( new App () );
            ...
        }
    }
}
```

In the Android project, call Toolkit.Init with the MainActivity and Bundle objects in the MainActivity class after the normal Forms.Init call:

```

namespace TextFileAsync.Droid
{
    ...
    public class MainActivity : global::Xamarin.Forms.Platform.Android.FormsApplicationActivity
    {
        protected override void OnCreate(Bundle bundle)
        {
            ...
            global::Xamarin.Forms.Forms.Init( this , bundle );
            Xamarin.FormsBook.Platform.Android.Toolkit.Init( this , bundle );
            LoadApplication( new App () );
        }
    }
}
```

In the three Windows platforms, call Toolkit.Init right after Forms.Init in the App.xaml.cs file:

```

namespace TextFileAsync.UWP
{
    ...
    sealed partial class App : Application
    {
        ...
        Xamarin.Forms.Forms.Init();
        Xamarin.FormsBook.Platform.UWP.Toolkit.Init();
        ...
    }
}
```

With that overhead out of the way, the actual writing of the application can begin. The XAML file

para `TextFileAsyncPage` es lo mismo que `TextFileTryoutPage`, pero el archivo de código subyacente debe ser formado para trabajar con el archivo asíncrono métodos de E / S. Todas las excepciones que podrían ocurrir en el archivo de E / S de funciones debe estar atrapado aquí, lo que significa que cualquier método que puede lanzar una excepción debe estar en una tratar bloquear junto con el esperar operador:

```
p\u00f3blico clase parcial TextFileAsyncPage : Pagina de contenido
{
    FileHelper fileHelper = nuevo FileHelper();

    p\u00f3blico TextFileAsyncPage ()
    {
        InitializeComponent ();

        RefreshListView ();
    }

    vacio asincrono OnSaveButtonClicked ( objeto remitente, EventArgs args)
    {
        saveButton.IsEnabled = falso ;

        cuerda nombre de archivo = filenameEntry.Text;

        Si ( esperar fileHelper.ExistsAsync (filename))
        {
            bool okResponse = esperar DisplayAlert ( "TextFileTryout",
                "Archivo " + Nombre +
                "Ya existe. Reemplazarlo?", ,
                "S\u00f1", "No" );

            Si (! OkResponse)
                regreso ;
        }

        cuerda errorMessage = nulo ;

        tratar
        {
            esperar fileHelper.WriteStringAsync (filenameEntry.Text, fileEditor.Text);
        }
        captura ( Excepcion Exc)
        {
            errorMessage = exc.Message;
        }

        Si (ErrorMessage == nulo )
        {
            filenameEntry.Text = "";
            fileEditor.Text = "";
            RefreshListView ();
        }
        m\u00e1s
        {
            esperar DisplayAlert ( "TextFileTryout", mensaje de error, "DE ACUERDO" );
        }
    }
}
```

```

        saveButton.IsEnabled = cierto ;
    }

    vacío asíncrono OnFileListViewItemSelected ( objeto remitente, SelectedItemChangedEventArgs args)
    {
        Si (Args.SelectedItem == nulo )
            regreso ;

        cuerda nombre de archivo = ( cuerda ) Args.SelectedItem;
        cuerda errorMessage = nulo ;

        tratar
        {
            fileEditor.Text = esperar fileHelper.ReadTextAsync (( cuerda ) Args.SelectedItem);
            filenameEntry.Text = nombre de archivo;
        }
        captura ( Excepción Exc )
        {
            errorMessage = exc.Message;
        }

        Si (ErrorMessage == nulo )
        {
            esperar DisplayAlert ( "TextFileTryout" , mensaje de error, "DE ACUERDO" );
        }
    }

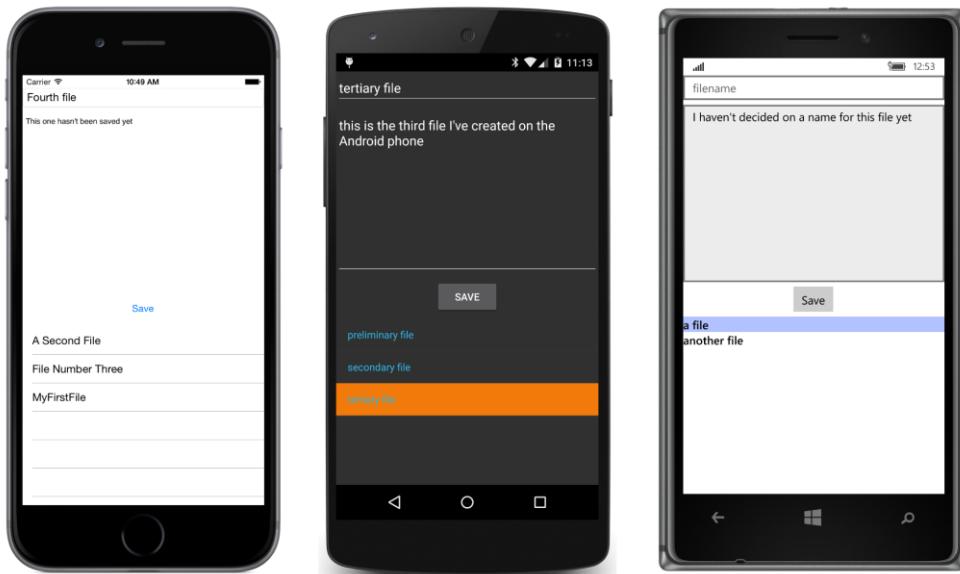
    vacío asíncrono OnDeleteMenuItemClicked ( objeto remitente, EventArgs args)
    {
        cuerda nombre de archivo = ( cuerda ) (( Opción del menú ) Remitente) .BindingContext;
        esperar fileHelper.DeleteAsync (filename);
        RefreshListView ();
    }

    vacío asíncrono RefreshListView ()
    {
        fileListView.ItemsSource = esperar fileHelper.GetFilesAsync ();
        fileListView.SelectedItem = nulo ;
    }
}

```

El resultado es que este código se estructura muy parecida a la anterior código que utiliza el archivo sincrónico funciones de E / S. Una diferencia, sin embargo, es que el OnSaveButtonClicked método desactiva el Salvar Botón al comenzar el procesamiento y después se vuelve a activar cuando todo está terminado. Esto es simplemente para evitar múltiples pulsaciones de la Salvar botón que podría causar la superposición de múltiples llamadas a FileIO.WriteAllText.

Aquí está el programa que se ejecuta en las tres plataformas:



Manteniéndolo en el fondo

Algunos de los FileHelper métodos en tiempo de ejecución de la implementación de Windows tienen múltiples esperar operadores para hacer frente a una serie de llamadas asíncronas. Esto tiene sentido: Cada paso en el proceso debe completar antes de que el siguiente paso se ejecuta. Sin embargo, una de las características de esperar es que se reanuda la ejecución en el mismo hilo que se invoca en lugar de la rosca de fondo. Esto a menudo es conveniente cuando se está obteniendo un resultado de actualizar la interfaz de usuario. Sin embargo, dentro de los métodos de la FileHelper implementaciones, esto no es necesario. Todo dentro del cuerpo de la Escribir-

TextAsync y ReadTextAsync métodos pueden ocurrir en una rosca secundaria.

los Tarea clase tiene un método denominado ConfigureAwait que puede controlar qué hilo esperar reanuda sucesivamente. Si pasa una falso argumento para ConfigureAwait, la tarea completada se reanudará en el mismo subproceso de trabajo utilizado para implementar la función. Si desea utilizar este en el FileHelper código, tendrá que convertir el IAsyncAction y IAsyncOperation objetos devueltos por los métodos de Windows en tiempo de ejecución de tareas mediante el uso de AsTask y luego llamar ConfigureAwait en ese Tarea objeto.

Por ejemplo, así es como el WriteTextAsync y ReadTextAsync métodos se implementan en el existente **Xamarin.FormsBook.Platform.WinRT** proyecto:

```
espacio de nombres Xamarin.FormsBook.Platform.WinRT
{
    clase FileHelper : IFileHelper
    {
        ...
        asíncrono pública Tarea WriteTextAsync ( cuerda nombre del archivo, cuerda texto)
        {
            StorageFolder carpetalocal = Datos de la aplicación .CurrentLocalFolder;
```

```
IStorageFile storageFile = esperar localFolder.CreateFileAsync (nombre de archivo,
    CreationCollisionOption .Remplaza el existente);
esperar FileIO .WriteTextAsync (storageFile, texto);
}

asíncrono pública Tarea < cuerda > ReadTextAsync ( cuerda nombre del archivo)
{
    StorageFolder carpetalocal = Datos de la aplicación .Current.LocalFolder;
    IStorageFile storageFile = esperar localFolder.GetFileAsync (filename);
    esperar volver FileIO .ReadTextAsync (storageFile);
}

...
}

}
```

Estos métodos tienen dos esperar operadores de cada uno. Para hacer estos métodos ligeramente más eficiente, puede utilizar AsTask y ConfigureAwait para cambiarlos a las siguientes:

```
espacio de nombres Xamarin.FormsBook.Platform.WinRT
{
    clase FileHelper : IFileHelper
    {
        ...
        asíncrono pública Tarea WriteTextAsync ( cuerda nombre del archivo, cuerda texto)
        {
            StorageFolder carpetalocal = Datos de la aplicación .Current.LocalFolder;
            IStorageFile storageFile = esperar localFolder.CreateFileAsync (nombre de archivo,
                CreationCollisionOption .Remplaza el existente).
                AsTask () .ConfigureAwait ( falso );
            esperar FileIO .WriteTextAsync (storageFile, texto) .AsTask () .ConfigureAwait ( falso );
        }

        asíncrono pública Tarea < cuerda > ReadTextAsync ( cuerda nombre del archivo)
        {
            StorageFolder carpetalocal = Datos de la aplicación .Current.LocalFolder;
            IStorageFile storageFile = esperar localFolder.GetFileAsync (nombre de archivo).
                AsTask () .ConfigureAwait ( falso );
            esperar volver FileIO .ReadTextAsync (storageFile) .AsTask () .ConfigureAwait ( falso );
        }
    }
}
```

Ahora los métodos siguientes a la primera esperar operador ejecuta en subprocesos de trabajo, y esperar no necesita cambiar de nuevo al hilo de interfaz de usuario sólo para continuar con el método. El interruptor de nuevo a la rosca UserInterface se produce cuando esperar se utiliza para llamar a estos métodos desde TextFileAsyncPage.

Es posible que desee restringir esta técnica a las funciones fundamentales de la biblioteca, o de código en sus clases de página que contienen una serie de esperar operadores que no tienen acceso a los objetos de interfaz de usuario. La técnica no hace tanto sentido para las funciones que contienen sólo una esperar operador que se llaman desde el subproceso de interfaz de usuario, ya que el interruptor de nuevo al hilo de interfaz de usuario tiene que ocurrir en cualquier momento, y si no se produce en la función de la biblioteca, se producirá en el código que llama a la biblioteca

función.

No bloquee el hilo de interfaz de usuario!

A veces, hay una tentación para evitar la molestia de Continua con o incluso la menor molestia de esperar simplemente bloqueando el hilo de interfaz de usuario hasta que se completa un proceso de fondo. Tal vez usted sabe que el proceso en segundo plano se completa muy rápidamente y no hay mucho que el usuario puede hacer de todos modos hasta que termine. ¿Cuál es el problema?

No lo haga! No sólo es de mala educación para el usuario, pero puede introducir errores sutiles en su aplicación.

Tomemos un ejemplo: En el archivo de código subyacente de `TextFileAsyncPage`, el `OnFileListView-ItemSelected` manejador tiene el siguiente código para leer el archivo y establecer el contenido de la Editor:

```
fileEditor.Text = esperar fileHelper.ReadTextAsync (( cuerda ) Args.SelectedItem);
```

Es posible que haya descubierto, tal vez por accidente o tal vez por el experimento, que en una declaración como esta, puede dejar de lado el esperar operador y sólo el acceso al Resultado propiedad de la Tarea `<string>`

objeto devuelto desde `ReadTextAsync`. Ese Resultado La propiedad es el contenido del archivo que se lee:

```
fileEditor.Text = fileHelper.ReadTextAsync (( cuerda ) Args.SelectedItem) .Result;
```

El código parece estar bien, y que incluso podría trabajar. Pero la forma en que funciona no es bueno. Esta declaración será bloquear el hilo de interfaz de usuario hasta que el `ReadTextAsync` método ha completado y Resultado está disponible. La interfaz de usuario no responderá durante este tiempo.

Por otra parte, si no se ha utilizado `ConfigureAwait (false)` en la implementación de `ReadTextAsync` en `FileHelper`, entonces eso `ReadTextAsync` método requerirá el cambio a la rosca de interfaz de usuario para la reanudación de la ejecución después de cada esperar operador. Pero cuando se trata de cambiar de nuevo al hilo `UserInterface`, el hilo de interfaz de usuario no estará disponible porque está siendo bloqueado en el `ReadTextAsync` llamar `TextFileAsyncPage`, y un clásico de los resultados de estancamiento. El programa simplemente dejan de ejecutarse en su totalidad.

La regla es simple: Uso Continua con o esperar con cada método asíncrono.

Sus propios métodos awaitable

Aparte de acceder a los archivos a través de Internet o desde el sistema de archivos local, las aplicaciones tienen a veces la necesidad de realizar largas operaciones propias. Estas operaciones se deben ejecutar en segundo plano en las discusiones secundarias de ejecución. Mientras que en la actualidad hay varias maneras de hacer esto, es mejor (y sin duda más fácil) para utilizar el mismo modelo asíncrono basado en tareas que se utiliza dentro de `Xamarin.Forms` y otros .NET entornos gráficos y definir sus propios métodos asíncronos al igual que los otros en estos ambientes.

La forma más fácil de ejecutar algún código en un subprocesso de trabajo es con el `Task.Run` y `Task.Run <T>`

métodos estáticos. El argumento es una Acción objeto, expresa generalmente como una función lambda, y el valor de retorno es una Tarea. El cuerpo de la función lambda se ejecuta en un subprocesso de trabajo del grupo de subprocessos, que (si desea utilizar el grupo de subprocessos mismo) es accesible a través de la ThreadPool clase. Se puede utilizar el esperar operador directamente con Task.Run:

```
esperar Tarea .Llevar () =>
{
    // El código que se ejecuta en un subprocesso de fondo.
});
```

Aunque se puede utilizar Task.Run por sí mismo con otro código, por lo general se utiliza para construir métodos asíncronos. Por convención, un método asíncrono tiene un sufijo de Asíncrono. El método devuelve un Tarea objeto (si el método no devuelve ningún valor u objeto) o una Tarea <T> objeto (si lo hace volver algo).

He aquí cómo usted puede crear un método asíncrono que devuelve Tarea:

```
Tarea MyMethodAsync (...)

{
    // Tal vez algún código de inicialización.
    regreso Tarea .Llevar () =>
    {
        // El código que se ejecuta en un subprocesso de fondo.
    });
}
```

los Task.Run método devuelve una Tarea objeto de que su método también devuelve. los Acción argumento para Task.Run Puede utilizar cualquier argumentos que se pasan a la MyMethodAsync, pero no se debe definir ningún argumento usando árbito o fuera. También, tenga cuidado con cualquier tipo de referencia se pasan a MyMethodAsync. Se puede acceder tanto desde dentro del código asíncrono y desde fuera del método, por lo que podrían tener que implementar la sincronización de manera que el objeto no se accede simultáneamente a partir de dos hilos.

El código dentro de la Task.Run llamada puede llamar a sí mismo usando métodos asíncronos esperar, pero en ese caso tendrá que bandera de la función lambda pasado a Task.Run con asíncrono:

```
regreso Tarea .Correr( asíncrono ) =>
{
    // El código que se ejecuta en un subprocesso de fondo.
});
```

Si el método asíncrono vuelve algo, se definirá el método que utiliza la forma genérica de Tarea y la forma genérica de Task.Run:

```
Tarea < Algun tipo > MyMethodAsync (...)

{
    // Tal vez algún código de inicialización.
    regreso Tarea .Llevar < Algun tipo > () =>
    {
        // El código que se ejecuta en un subprocesso de fondo.
        regreso anInstanceOfSomeType;
    });
}
```

}

El valor u objeto devuelto por la función lambda se convierte en el Resultado propiedad de la Tarea <T> objeto devuelto desde Task.Run y de su método.

Si necesita tener más control sobre el proceso en segundo plano, puede utilizar TaskFactory.StartNew más bien que Task.Run para definir el método asíncrono.

Hay algunas variaciones sobre el básico Task.Run patrones, como se verá en los siguientes varios programas. Estos programas calcular y mostrar el famoso conjunto de Mandelbrot.

El conjunto de Mandelbrot básica

El matemático francés y americano de origen polaco-Benoit Mandelbrot (1924-2010) es el más conocido por su trabajo relacionado con las superficies auto-similares complejas que llamó *fractales*. Entre sus trabajos que implican fractales era una investigación sobre una fórmula recursiva que genera una imagen del fractal que ahora se conoce como el *de Mandelbrot*.

El conjunto de Mandelbrot se representa gráficamente en el plano complejo, donde cada coordenada es un número complejo de la forma:

$\bullet = \bullet + \bullet$

La parte real X se representa gráficamente a lo largo del eje horizontal con valores negativos a los valores de izquierda y positiva a la derecha. La parte imaginaria y se representa gráficamente en el eje vertical, pasando de valores negativos en la parte inferior para valores positivos subiendo.

Para calcular el conjunto de Mandelbrot, empezar por tomar cualquier punto en este plano y lo llaman *do*, e inicializar z a cero:

$\bullet = \bullet + \bullet$

$\bullet = 0$

Ahora realice la siguiente operación recursiva:

$\bullet \leftarrow \bullet^2 + \bullet$

El resultado será bien divergen hasta el infinito o no será. Si z hace no divergir a infinito, entonces *do* se dice que es un miembro del conjunto de Mandelbrot. De lo contrario, no es un miembro del conjunto de Mandelbrot.

Es necesario realizar este cálculo para cada punto de interés en el plano complejo. En general, los resultados se dibujan en un mapa de bits, lo que significa que cada píxel en el mapa de bits corresponde a un complejo en particular de coordenadas. En su versión más simple, los puntos que pertenecen al conjunto de Mandelbrot son de color negro y otros píxeles son de color blanco.

Para algunos números complejos, es fácil determinar si el punto pertenece al conjunto de Mandelbrot. Por ejemplo, el número complejo $(0 + 0 y)$ obviamente, pertenece al conjunto de Mandelbrot, y se puede establecer rápidamente que $(1 + 0 y)$ no. Pero, en general, es necesario realizar el cálculo recursivo.

Y porque este es un fractal, no se puede tomar atajos. Por ejemplo, si usted sabe que los dos valores d_1 y d_2 pertenecer al conjunto de Mandelbrot, no se puede asumir que todos los puntos entre esos dos puntos pertenecen al conjunto de Mandelbrot también. Es una característica fundamental de un fractal de desafiar a la interpolación.

¿Cuántas iteraciones del cálculo recursivo se necesita para llevar a cabo antes de que pueda asegurarse que el número complejo particular que hace o no pertenece al conjunto de Mandelbrot? Resulta que si el valor absoluto de z en el cálculo recursivo vez se convierte en 2 o mayor, entonces los valores finalmente se desviaran hacia el infinito y el punto no pertenece al conjunto de Mandelbrot. (El valor absoluto de un número complejo también se conoce como la *magnitud* del número; se puede calcular como la raíz cuadrada de la suma de los cuadrados de la X y y valores, que es el teorema de Pitágoras.)

Sin embargo, si después de un cierto número de iteraciones el cálculo recursivo todavía no ha alcanzado una magnitud de 2, no hay garantía de que no va a divergir con iteraciones repetidas. Por esta razón, los conjuntos de Mandelbrot son muy intensivas en computación, y es ideal para las discusiones secundarias de ejecución.

los **MandelbrotSet** programa demuestra cómo se hace esto. Para representar la imagen, el programa hace uso de la **BmpMaker** clase (introducido en el Capítulo 13, “Los mapas de bits”) de la **Xamarin.FormsBook.Toolkit** biblioteca. Esta biblioteca también contiene la siguiente estructura para representar un número complejo:

```
espacio de nombres Xamarin.FormsBook.Toolkit
{
    // En su mayoría un subconjunto de System.Numerics.Complex.
    struct pública Complejo : IEquatable < Complejo >, IFormattable
    {
        bool gotMagnitude, gotMagnitudeSquared;
        doble magnitud, magnitudeSquared;

        público Complejo( doble real, doble imaginaria): esta()
        {
            Real = real;
            Imaginario = imaginario;
        }

        pública doble {real conjunto privado ; obtener ; }

        pública doble {imaginaria conjunto privado ; obtener ; }

        // MagnitudeSquare y magnitud calculada sobre la demanda y se guarda.
        pública doble MagnitudeSquared
        {
            obtener
            {
                Si (GotMagnitudeSquared)
                {
                    regreso magnitudeSquared;
                }

                magnitudSquared = Real * real + Imaginario Imaginario;
                gotMagnitudeSquared = cierto ;
                regreso magnitudSquared;
            }
        }
    }
}
```

```
}

pública doble Magnitud
{
    obtener
    {
        Si (GotMagnitude)
        {
            regreso magnitud;
        }

        = magnitud Mates.Sqrt (magnitudeSquared);
        gotMagnitude = cierto ;
        regreso magnitud;
    }
}

public static Complejo operador + ( Complejo izquierda, Complejo derecho)
{
    return new Complejo (Left.Real + right.Real, left.Imaginary + right.Imaginary);
}

public static Complejo operador - ( Complejo izquierda, Complejo derecho)
{
    return new Complejo (Left.Real - right.Real, left.Imaginary - right.Imaginary);
}

public static Complejo operador * ( Complejo izquierda, Complejo derecho)
{
    return new Complejo (* Left.Real right.Real - left.Imaginary * right.Imaginary,
                         left.Real * + right.Imaginary left.Imaginary * right.Real);
}

operador public static bool == ( Complejo izquierda, Complejo derecho)
{
    regreso left.Real == right.Real && left.Imaginary == right.Imaginary;
}

operador public static bool != ( Complejo izquierda, Complejo derecho)
{
    regreso ! (izquierda == derecha);
}

operador implícita public static Complejo ( doble valor)
{
    return new Complejo (Valor, 0);
}

operador implícita public static Complejo ( En t valor)
{
    return new Complejo (Valor, 0);
}

público int anulación GetHashCode ()
```

```

{
    regreso Real.GetHashCode () + Imaginary.GetHashCode ();
}

público bool anulación es igual a ( Objeto valor)
{
    regreso Real.Equals ((( Complejo ) Valor) .real) &&
        Imaginary.Equals ((( Complejo ) Valor) .Imaginary);
}

public bool es igual a ( Complejo valor)
{
    regreso Real.Equals (valor) && Imaginary.Equals (valor);
}

público cadena de anulación Encadenar()
{
    regreso Cuerda .Formato ("{0} {1} {2}" , Real,
        RealImaginaryConnector (imaginario),
        Mates .Abs (imaginario));
}

public string Encadenar( cuerda formato)
{
    regreso Cuerda .Formato ("{0} {1} {2}" , Real.ToString (formato),
        RealImaginaryConnector (imaginario),
        Mates .Abs (imaginario) .ToString (formato));
}

public string Encadenar( IFormatProvider formatProvider)
{
    regreso Cuerda .Formato ("{0} {1} {2}" , Real.ToString (formatProvider),
        RealImaginaryConnector (imaginario),
        Mates .Abs (imaginario) .ToString (formatProvider));
}

public string Encadenar( cuerda formato, IFormatProvider formatProvider)
{
    regreso Cuerda .Formato ("{0} {1} {2}" , Real.ToString (formato, formatProvider),
        RealImaginaryConnector (imaginario),
        Mates .Abs (imaginario) .ToString (formato, formatProvider));
}

cuerda RealImaginaryConnector ( doble valor)
{
    regreso Mates .sign (valor)> 0? "+" : "\u2013";
}
}
}

A medida que el comentario en la parte superior indica, se trata de principalmente un subconjunto de la Complejo estructura en el .NET System.Numerics espacio de nombres, que por desgracia no está disponible para una biblioteca de clases portátil en un proyecto Xamarin.Forms. los Encadenar métodos en este Complejo estructura de trabajo un poco diferente, sin embargo,
```

y el original Complejo la estructura no tiene una MagnitudeSquared propiedad. UN Magnitud-squared propiedad es útil para el cálculo de Mandelbrot: Verificando la Magnitud la propiedad es menor que 2 es el mismo que el de comprobar si el MagnitudeSquared la propiedad es menor que 4, pero sin el cálculo de la raíz cuadrada.

los **MandelbrotSet** programa tiene el siguiente archivo XAML:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " MandelbrotSet.MandelbrotSetPage " >

    < ContentPage.Padding >
        < OnPlatform x: TypeArguments = " Espesor "
            iOS = " 0, 20, 0, 0 " />
    </ ContentPage.Padding >

    < StackLayout >
        < Cuadricula VerticalOptions = " FillAndExpand " >
            < ContentView Relleno = " 10, 0 "
                VerticalOptions = " Centrar " >
                < ActivityIndicator x: Nombre = " activityIndicator " />
            </ ContentView >

            < Imagen x: Nombre = " imagen " />
        </ Cuadricula >

        < Botón x: Nombre = " calculateButton "
            Texto = " Calcular "
            Tamaño de fuente = " Grande "
            HorizontalOptions = " Centrar "
            hecho clic = " OnCalculateButtonClicked " />
    </ StackLayout >
</ Página de contenido >
```

los **ActivityIndicator** informa al usuario que el programa está ocupado con el trabajo en segundo plano. los Estoy-años y que el elemento **ActivityIndicator** compartir una sola célula Cuadricula de manera que la **ActivityIndicator** puede ser más hacia el centro vertical de la pantalla y luego se cubren cuando aparece el mapa de bits. En la parte inferior es una Botón para iniciar el cálculo.

El archivo de código subyacente a continuación comienza por definir varias constantes. Los primeros cuatro constantes se relacionan con el mapa de bits que el programa construye para mostrar la imagen del conjunto de Mandelbrot. A lo largo de este ejercicio, estos mapas de bits siempre será cuadrado, pero el código en sí es más generalizada y deben ser capaces de adaptarse a las dimensiones rectangulares.

los **centrar campo** es el Complejo punto que corresponde al centro del mapa de bits, mientras que la **tamaño campo** indica la medida de las coordenadas reales e imaginarios en los mapas de bits. estos particular, **center** y **tamaño** campos implican que las coordenadas reales varían desde -2 a la izquierda del mapa de bits a 0,5 a la derecha, y las coordenadas imaginarias van desde -1,25 en la parte inferior a 1,25 en la parte superior. los **pixelWidth** y **pixelHeight** valores indican el ancho y la altura del mapa de bits en píxeles. los **iterativo** ciones campo es el número máximo de iteraciones de la fórmula recursiva antes de que el programa asume que el punto pertenece al conjunto de Mandelbrot:

```

pública clase parcial MandelbrotSetPage : Pagina de contenido
{
    estático solo lectura Complejo centro = nuevo Complejo (-0,75, 0);
    estático solo lectura tamaño size = nuevo tamaño (2,5, 2,5);
    const int PixelWidth = 1,000;
    const int pixelHeight = 1,000;
    const int iteraciones = 100;

    pública MandelbrotSetPage ()
    {
        InitializeComponent ();
    }

    vacío asíncrono OnCalculateButtonClicked ( objeto remitente, EventArgs args)
    {
        calculateButton.IsEnabled = falso ;
        activityIndicator.IsRunning = cierto ;

        BmpMaker bmpMaker = nuevo BmpMaker (PixelWidth, pixelHeight);
        esperar CalculateMandelbrotAsync (bmpMaker);
        image.Source = bmpMaker.Generate ();

        activityIndicator.IsRunning = falso ;
    }

    Tarea CalculateMandelbrotAsync ( BmpMaker bmpMaker)
    {
        regreso Tarea .Llevar () =>
        {
            para ( En t fila = 0; fila <pixelHeight; fila++)
            {
                doble y = center.Imaginary - size.Height / 2 + fila * size.Height / pixelHeight;

                para ( En t col = 0; Col <PixelWidth; col++)
                {
                    doble x = center.Real - Tamano.Width / 2 + col * Tamano.Width / PixelWidth;
                    Complejo c = nuevo Complejo (X, y);
                    Complejo z = 0;
                    En t iteración = 0;

                    hacer
                    {
                        z = z * z + c;
                        iteración++;
                    }
                    mientras (iteración <iteraciones && z.MagnitudeSquared <4);

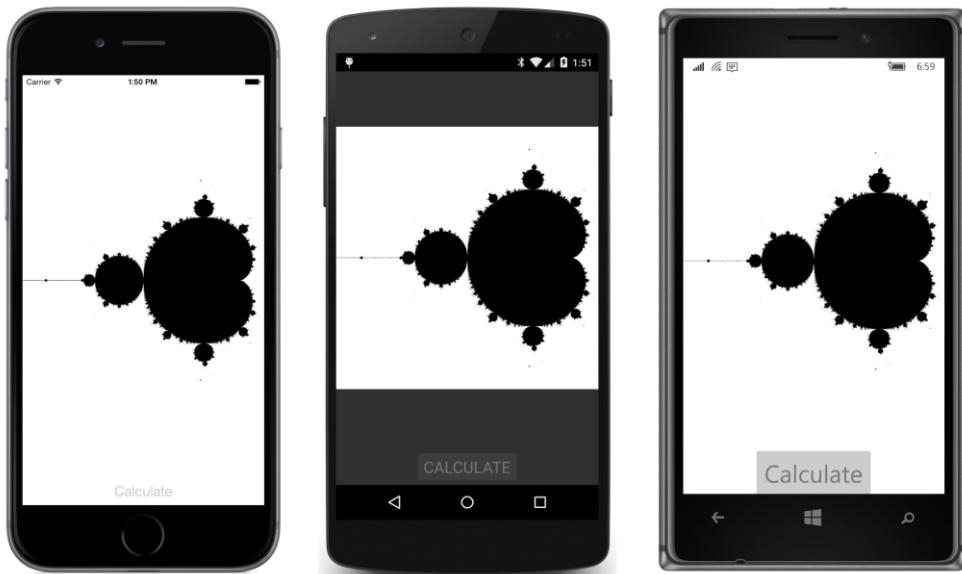
                    bool isMandelbrotSet = iteración == iteraciones;
                    bmpMaker.SetPixel (fila, col, isMandelbrotSet? Color .black: Color .Blanco);
                }
            }
        });
    }
}

```

los OnCalculateButtonClicked manejador se marca como asíncrono. Se inicia mediante la desactivación de la Botón para evitar múltiples cálculos simultáneos e inicia el ActivityIndicator monitor. A continuación, crea una BmpMaker objeto con el tamaño de píxel deseado y lo pasa a CalculateMandelbrotAsync. Cuando se termina ese método, la hecho clic guía continúa estableciendo el mapa de bits a la Imagen objetar y apagar el ActivityIndicator. Los Botón No se vuelve a habilitar.

La función lambda se pasa a la Task.Run método de bucle a través de las filas y columnas del mapa de bits creados por BmpMaker, y para cada píxel, se calcula un número complejo do desde el X y y los valores de coordenadas. El pequeño do-while bucle continúa hasta que se alcanza el número máximo de iteraciones o la magnitud es 2 o mayor. En ese momento, un píxel se puede ajustar a negro o blanco.

Después de pulsar el botón, el teléfono puede tardar un minuto más o menos a recorrer todos los píxeles, pero luego se va a ver la imagen clásica:



Hay un poco de peligro en la forma en que el CalculateMandelbrotAsync método está estructurado. Se aprobó una BmpMaker objetar que el hilo de fondo se llena de píxeles, pero el hilo principal también tiene acceso a esta BmpMaker objeto. Si este objeto se guarda como un campo, el hilo principal podría contener un código que altere o define los píxeles como el subproceso en segundo plano está funcionando. Que probablemente sería un error, por supuesto, pero en general se puede hacer que sus métodos asíncronos más a prueba de balas si los argumentos están restringidos a los tipos de valor en lugar de los tipos de referencia. No se preocupe demasiado si eso no es bastante posible o conveniente, pero en la siguiente versión del programa, la CalculateMandelbrotAsync método en sí crear la BmpMaker objeto y devolverlo.

marcando el progreso

Como usted ha descubierto, sin duda, es un tanto desconcertante para presionar el **Calcular** botón de **MandelbrotSet** y esperar a que el mapa de bits en aparecer. No hay indicación alguna de qué tan avanzado el programa ha llegado a completar el trabajo, o cuánto tiempo tendrá que esperar.

Si es posible, métodos asíncronos deben informar el progreso. Estoy seguro de que te permita algo a sí mismo para hacer el trabajo, pero hay una forma estándar de informar sobre el progreso de los métodos que devuelven Tarea objetos. Esto implica la **IProgress** <T> interfaz y el **El progreso** <T> clase que implementa la interfaz, ambos de los cuales están definidos en el Sistema espacio de nombres. **IProgress** se define así:

```
pùblico interfaz IProgress <T>
{
    vacio Informe (valor de T);
}
```

Para hacer uso de este servicio, se define un argumento a su método asíncrono de tipo Yo soy profesional-
Gress. El método asíncrono continuación, llama periódicamente Informe como lo está haciendo el trabajo en segundo plano. En general, T es cualquiera
En t, en cuyo caso los valores pasados a Informe generalmente varían de 1 a 100, o
doble, para los valores que van de 0 a 1. Es su elección. Por coherencia con las Xamarin.Forms Pro-
gressBar, doble valores de 0 a 1 son ideales.

El código que llama al método asíncrono instancia un Progreso objeto y pasa a su constructor una función lambda que se llama
cuando las llamadas a métodos asíncronos Informe. (O se puede adjuntar un controlador a la Progreso objetos ProgressChanged evento).
Aunque Informe se llama en un subproceso en segundo plano, el controlador de función lambda o evento se llama en el subproceso que
crea una instancia del
Progreso objeto, lo que significa que la función lambda o controlador de eventos pueden acceder de forma segura los objetos UserInterface.

El archivo XAML para el **MandelbrotProgress** programa es el mismo que el archivo XAML anterior excepto que una Barra de
progreso ha reemplazado a la ActivityIndicator:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " MandelbrotProgress.MandelbrotProgressPage " >

< ContentPage.Padding >
    < OnPlatform x: TypeArguments = " Espesor "
        iOS = " 0, 20, 0, 0 " />
</ ContentPage.Padding >

< StackLayout >
    < Cuadrícula VerticalOptions = " FillAndExpand " >
        < ContentView Relleno = " 10, 0 "
            VerticalOptions = " Centrar " >
            < Barra de progreso x: Nombre = " barra de progreso " />
        </ ContentView >

        < Imagen x: Nombre = " imagen " />
    </ Cuadrícula >

```

```

< Botón x: Nombre = " calculateButton "
    Texto = " Calcular "
    Tamaño de fuente = " Grande "
    HorizontalOptions = " Centrar "
    hecho clic = " OnCalculateButtonClicked " />

</ StackLayout >
</ Pagina de contenido >

```

El archivo de código subyacente es muy similar, excepto que una Progreso objeto nombrado progressReporter se define como un campo y el constructor de instancia con una función lambda que simplemente establece el argumento de la Progreso propiedad de la Barra de progreso. Esta Progreso objeto se pasa a la calculateMandelbrotAsync método, que en esta nueva versión tiene ahora la responsabilidad de crear y devolver el BmpMaker objeto:

```

público clase parcial MandelbrotProgressPage : Pagina de contenido
{
    estático solo lectura Complejo centro = nuevo Complejo (-0,75, 0);
    estático solo lectura tamaño size = nuevo tamaño (2,5, 2,5);
    const int PixelWidth = 1,000;
    const int pixelHeight = 1,000;
    const int iteraciones = 100;

    Progreso < doble > ProgressReporter;

    público MandelbrotProgressPage ()
    {
        InitializeComponent ();

        progressReporter = nuevo Progreso < doble > (( doble valor) =>
        {
            ProgressBar.progress = valor;
        });
    }

    vacío asíncrono OnCalculateButtonClicked ( objeto remitente, EventArgs args)
    {
        // Configurar la interfaz de usuario para un proceso en segundo plano.
        calculateButton.IsEnabled = falso ;

        // genera el conjunto de Mandelbrot en un mapa de bits.
        BmpMaker bmpMaker = esperar CalculateMandelbrotAsync (progressReporter);
        image.Source = bmpMaker.Generate ();
    }

    Tarea < BmpMaker > CalculateMandelbrotAsync ( IProgress < doble > Curso)
    {
        regreso Tarea .Llevar < BmpMaker > (() =>
        {
            BmpMaker bmpMaker = nuevo BmpMaker (PixelWidth, pixelHeight);

            para ( En t fila = 0; fila < pixelHeight; fila++)
            {
                doble y = center.Imaginary - size.Height / 2 + fila * size.Height / pixelHeight;

```

```

// memoria los progresos.

Informe de progreso(( double ) De fila / pixelHeight);

para ( En t col = 0; Col <PixelWidth; col++)
{
    doble x = center.Real - Tamano.Width / 2 + col * Tamano.Width / PixelWidth;

    Complejo c = nuevo Complejo (x, y);
    Complejo z = 0;

    En t iteración = 0;
    bool isMandelbrotSet = falso ;

    Si ((C - nuevo Complejo (-1, 0)). MagnitudeSquared <1,0 / 16)
    {
        isMandelbrotSet = cierto ;
    }
    más
    {
        hacer
        {
            z = z * z + c;
            iteración++;
        }
        mientras (iteración <iteraciones && z.MagnitudeSquared <4);

        isMandelbrotSet = iteración == iteraciones;
    }
    bmpMaker.SetPixel (fila, col, isMandelbrotSet? Color .black: Color .Blanco);
}

regreso bmpMaker;
});
```

El método asíncrono informa sobre su progreso con cada nueva fila:

Informe de progreso((doble) De fila / pixelHeight);

Cuidado con: Usted no quiere reportar el progreso con tanta frecuencia que disminuye la velocidad del método! Un centenar de llamadas a la Informe método durante toda la operación es un montón, y es probable que pueda reducir ese número considerablemente antes de la Barra de progreso comienza mirando nervioso.

Si se presta mucha atención a la Barra de progreso en **MandelbrotProgress**, verá que se mueve rápido en el inicio y luego se ralentiza. El área del problema es la gran cardioide y, en menor medida, el círculo a su izquierda, que constituye la mayor parte del conjunto de Mandelbrot. Para los puntos dentro de estas áreas, el cálculo recursivo debe ejecutar para el número máximo de iteraciones antes de que se identifique el punto como un miembro del conjunto. Este nuevo método intenta reducir el trabajo algo mediante la detección de cuando el punto se encuentra dentro del círculo. El centro de este círculo es el punto -1 complejo, y el radio es 1/4:

```
Si ((C - nuevoComplejo (-1, 0)). MagnitudeSquared <1,0 / 16)
{
    isMandelbrotSet = cierto;
```

```
}
```

Pero la cardioide es un objeto más complejo (aunque eso también puede ser identificado, ya que la próxima versión del programa demuestra).

Cuando el método asíncrono crea y rendimientos que BmpMaker objeto, el código para obtener ese objeto y establecer el mapa de bits para la Imagen objeto se reduce a sólo dos estados:

```
BmpMaker bmpMaker = esperar CalculateMandelbrotAsync (progressReporter);
image.Source = bmpMaker.Generate ();
```

Pero si dos afirmaciones son demasiados, tener en cuenta que esperar es más o menos sólo un operador ordinario y puede ser parte de una declaración más compleja:

```
image.Source = ( esperar CalculateMandelbrotAsync (progressReporter) Generar () );
```

Cancelación del trabajo

Los dos programas mostrados Mandelbrot medida en que existen con el único propósito de generar una sola imagen, por lo que es poco probable que usted desee cancelar ese trabajo una vez comenzado. Sin embargo, en el caso general, tendrá que proporcionar una facilidad para el usuario para rescatar a de largos trabajos en segundo plano.

Aunque es probable que pueda armar un poco de sistema de cancelación de su cuenta, el System.Threading espacio de nombres ya se ha cubierto con una clase llamada CancellationTokenFuente y una estructura nombrada CancellationToken.

Así es como funciona:

Un programa crea una CancellationTokenSource para su uso con un método asíncrono particular. Los CancellationTokenSource clase define una propiedad denominada Simbólico que devuelve una cancellationToken. Esta CancellationToken valor se pasa al método asíncrono. El método asíncrono llama periódicamente la IsCancellationRequested método de la CancellationToken.

Este método por lo general regresa falso.

Cuando el programa se quiere cancelar la operación asíncrona (probablemente en respuesta a alguna entrada del usuario), llama al Cancelar método de la CancellationTokenSource. La próxima vez que el método asíncrono llama al IsCancellationRequested método de la CancellationToken, el método devuelve cierto debido a una cancelación ha sido solicitada. El método asíncrono puede elegir la forma de detener la ejecución, tal vez con un simple regreso declaración.

Por lo general, sin embargo, se toma un enfoque diferente. En lugar de llamar a la IsCancellationRequested método de CancellationToken, el método asíncrono puede que simplemente llamar a la ThrowIfCancellationRequested método. Si se ha solicitado una cancelación, el método asíncrono deja de ejecutar elevando una OperationCanceledException.

Esto significa que el esperar el operador tiene que ser parte de una tratar bloque, pero como se ha visto, este es generalmente el caso cuando se trabaja con archivos, por lo que no agrega mucho código adicional, y el programa puede procesar una cancelación simplemente como otra forma de excepción.

los **MandelbrotCancellation** programa muestra esta técnica. El archivo XAML ahora tiene un segundo botón, la etiqueta "Cancelar", que está inicialmente desactivado:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " MandelbrotCancellation.MandelbrotCancellationPage " >

    < ContentPage.Padding >
        < OnPlatform x: TypeArguments = " Espesor "
            iOS = " 0, 20, 0, 0 " />
    </ ContentPage.Padding >

    < StackLayout >
        < Cuadrícula VerticalOptions = " FillAndExpand " >
            < ContentView Relleno = " 10, 0 "
                VerticalOptions = " Centrar " >
                < Barra de progreso x: Nombre = " barra de progreso " />
            </ ContentView >

            < Imagen x: Nombre = " imagen " />
        </ Cuadrícula >

        < Cuadrícula >
            < Botón x: Nombre = " calculateButton "
                Grid.Column = " 0 "
                Texto = " Calcular "
                Tamaño de fuente = " Grande "
                HorizontalOptions = " Centrar "
                hecho clic = " OnCalculateButtonClicked " />

            < Botón x: Nombre = " CancelButton "
                Grid.Column = " 1 "
                Texto = " Cancelar "
                Tamaño de fuente = " Grande "
                Está habilitado = " Falso "
                HorizontalOptions = " Centrar "
                hecho clic = " OnCancelButtonClicked " />
        </ Cuadrícula >
    </ StackLayout >
</ Página de contenido >
```

El archivo de código subyacente tiene ahora una más amplia `OnCalculateButtonClicked` método. Se inicia mediante la desactivación de la **Calcular** botón y permite al **Cancelar** botón. Se crea una nueva Cancelación-`TokenSource` objeto y pasa el `Simbólico` propiedad a `CalculateMandelbrotAsync`. Los `OnCancelButtonClicked` método es responsable de llamar `Cancelar` sobre el `CancellationTokenSource` objeto. Los `CalculateMandelbrotAsync` llama al método `ThrowIfCancellationRequested` método en la misma proporción que informa el progreso. La excepción es capturado por el `OnCalculateButtonClicked` método, que responde por reenabling la **Calcular** Botón para otro intento:

```
ímparo clase parcial MandelbrotCancellationPage : Página de contenido
{
    estatico solo lectura Complejo centro = nuevo Complejo (-0,75, 0);
    estatico solo lectura tamaño size = nuevo tamaño (2,5, 2,5);
    const int PixelWidth = 1,000;
```

```
const int pixelHeight = 1,000;
const int iteraciones = 100;

Progress < doble > ProgressReporter;
CancellationTokenSource cancelTokenSource;

pùblico MandelbrotCancellationPage ()
{
    InitializeComponent ();

    progressReporter = nuevo Progress < doble > (( doble valor) =>
    {
        ProgressBar.progress = valor;
    });
}

vacío asíncrono OnCalculateButtonClicked ( objeto remitente, EventArgs args)
{
    // Configurar la interfaz de usuario para un proceso en segundo plano.
    calculateButton.IsEnabled = falso ;
    cancelButton.IsEnabled = cierto ;

    cancelTokenSource = nuevo CancellationTokenSource ();

    tratar
    {
        // genera el conjunto de Mandelbrot en un mapa de bits.
        BmpMaker bmpMaker = esperar CalculateMandelbrotAsync (progressReporter,
                                                               cancelTokenSource.Token);

        image.Source = bmpMaker.Generate ();
    }
    captura ( OperationCanceledException )
    {
        calculateButton.IsEnabled = cierto ;
        ProgressBar.progress = 0;
    }
    captura ( Excepción )
    {
        // no debe ocurrir en este caso.
    }
    cancelButton.IsEnabled = falso ;
}

vacío OnCancelButtonClicked ( objeto remitente, EventArgs args)
{
    cancelTokenSource.Cancel ();
}

Tarea < BmpMaker > CalculateMandelbrotAsync ( IProgress < doble > Progreso,
                                                CancellationToken cancelToken)
{
    regreso Tarea .Llevar < BmpMaker > ( () =>
    {
```

```

BmpMaker bmpMaker = nuevo BmpMaker (PixelWidth, pixelHeight);

para ( En t fila = 0; fila <pixelHeight; fila++)
{
    doble y = center.Imaginary - size.Height / 2 + fila * size.Height / pixelHeight;

    // memoria los progresos.
    Informe de progreso(( doble ) De fila / pixelHeight);

    // Es posible cancelar.
    cancelToken.ThrowIfCancellationRequested ();

    para ( En t col = 0; Col <PixelWidth; col++)
    {
        doble x = center.Real - Tamano.Width / 2 + col * Tamano.Width / PixelWidth;
        Complejo c = nuevo Complejo (X, y);
        Complejo z = 0;
        En t iteración = 0;
        bool isMandelbrotSet = falso ;

        Si ((C - nuevo Complejo (-1, 0)). MagnitudeSquared <1,0 / 16)
        {
            isMandelbrotSet = cierto ;
        }
        // http://www.reenigne.org/blog/algorithm-for-mandelbrot-cardioid/
        else if (C.MagnitudeSquared * (8 * c.MagnitudeSquared - 3) <
3.0 / 32 - C.Real)
        {
            isMandelbrotSet = cierto ;
        }
        más
        {
            hacer
            {
                z = z * z + c;
                iteración++;
            }
            mientras (iteración <iteraciones && z.MagnitudeSquared <4);
            isMandelbrotSet = iteración == iteraciones;
        }
        bmpMaker.SetPixel (fila, col, isMandelbrotSet? Color .black: Color .Blanco);
    }
}
regreso bmpMaker;
}, CancelToken);
}
}

```

los CancellationToken También se pasa como segundo argumento a Task.Run. Esto no es necesario, sino que permite la Task.Run Método para saltar mucho trabajo si ya se ha solicitado la cancelación antes de que incluso se inicia.

Observe también que el código ahora salta a la gran cardioide. Un comentario hace referencia a una página web que se deriva la fórmula en caso de que desee comprobar los cálculos.

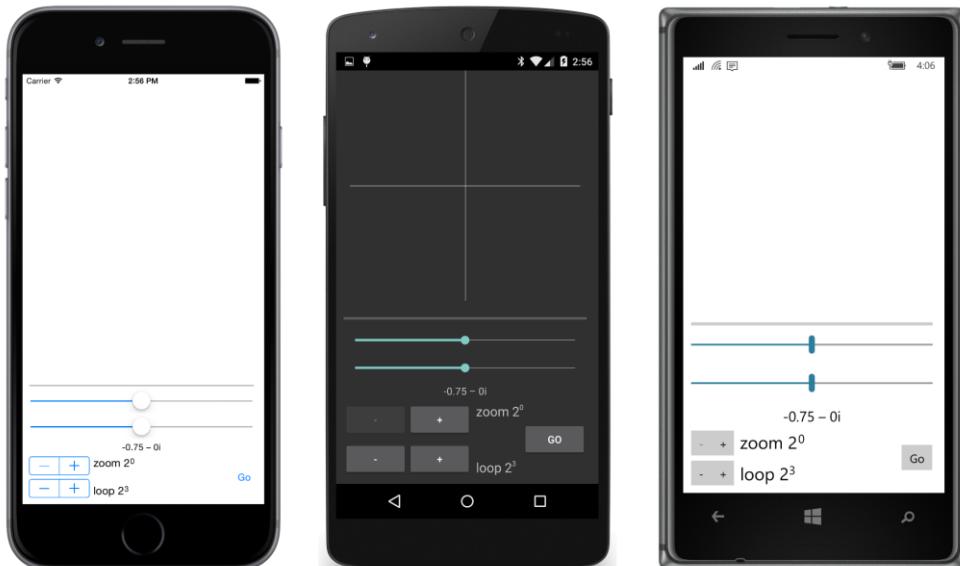
Un Mandelbrot MVVM

Aunque el conjunto de Mandelbrot blanco y negro es la imagen clásica, la mayoría de los programas de Mandelbrot píxeles de color que no están en el conjunto de Mandelbrot basado en el número de iteraciones necesarias para esa determinación. El penúltimo programa en este capítulo se llama **MandelbrotXF** -la XF prefijo significa Xamarin.Forms-colores y los píxeles de esa manera. El programa también permite hacer zoom sobre lugares específicos. Es una característica de un conjunto de Mandelbrot que la imagen sigue siendo interesante, no importa lo lejos que uno se acerque. Por desgracia, hay un límite práctico para el uso del zoom basado en la resolución de los números de punto flotante doubleprecision.

El programa posee una arquitectura utilizando principios MVVM, aunque después de ver la interfaz de usuario tanto extraño, y cómo las ofertas de ViewModel con ese usuario interfaz que te pueden cuestionar la sabiduría de esa decisión.

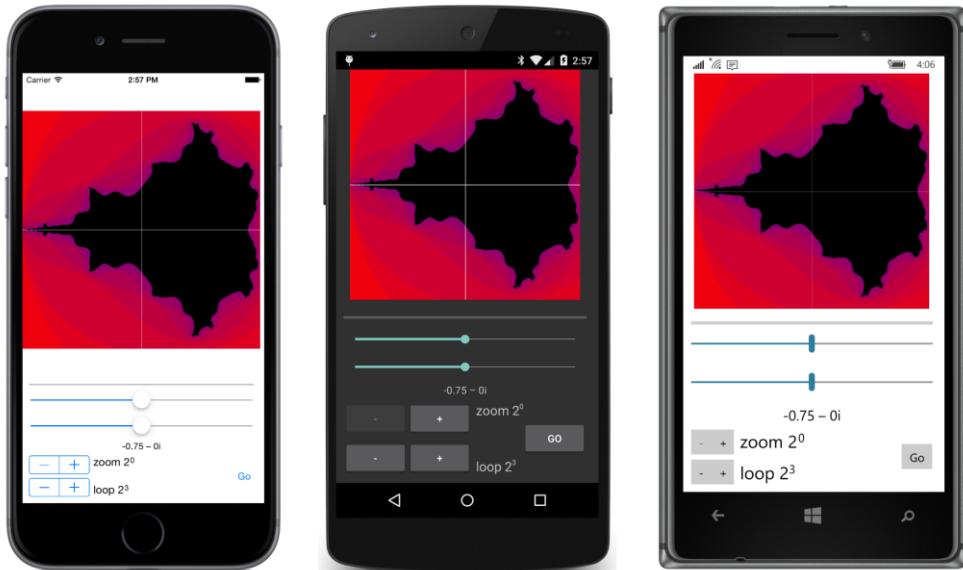
La interfaz de usuario impar de **MandelbrotXF** el resultado de una decisión de evitar cualquier código específico de la plataforma. En el momento de este programa fue escrito originalmente, Xamarin.Forms no apoyaron las operaciones táctiles, como arrastrar y pellizcos que podría haber sido útil para hacer zoom en un lugar determinado. En su lugar, toda la interfaz de usuario del programa es implementado con dos deslizador elementos, dos paso a paso elementos, dos Botón elementos, una ProgressBar, y efectos visuales a cabo con BoxView.

La primera vez que ejecute el programa, esto es lo que verá:

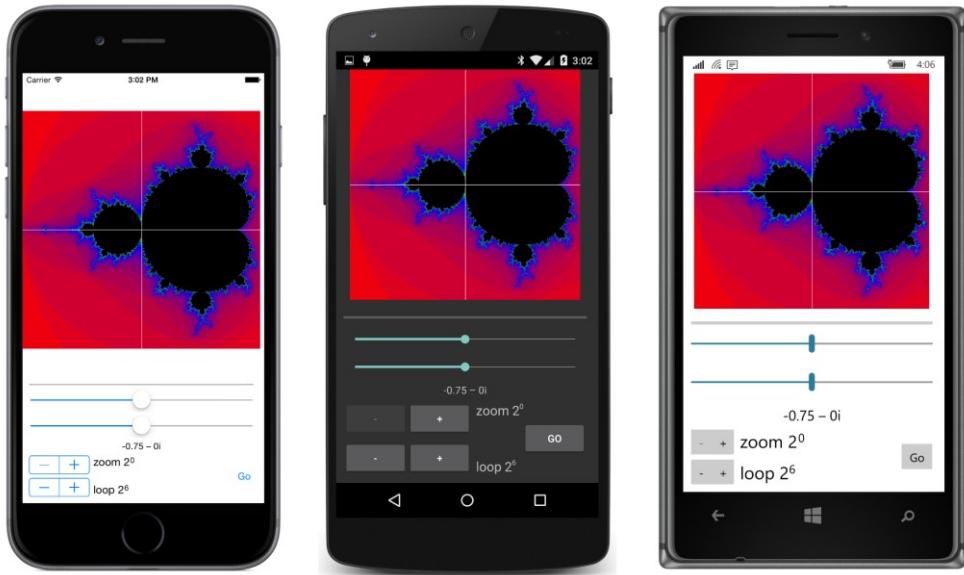


El punto de mira -que blancos no se presenten contra el fondo blanco de los IOS en blanco y de Windows Mobile 10 pantallas se desvanecen en el transcurso de 10 segundos, por lo que no van a oscurecer las imágenes bonitas que pronto estará admirando, pero se puede traer de vuelta mediante la manipulación de cualquiera de los reguladores o los motores paso a paso.

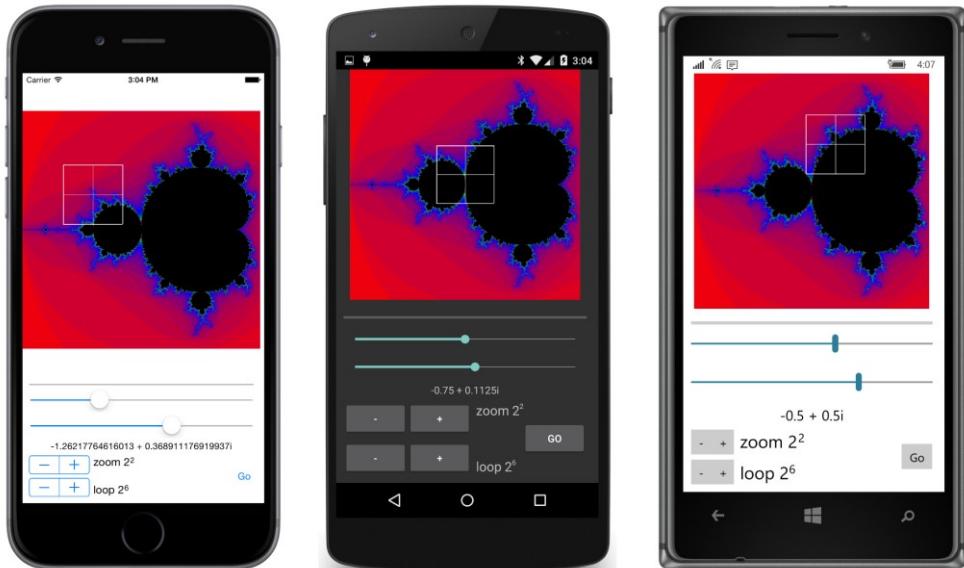
Pero lo primero que querrá hacer es presionar el Ir botón. El botón se reemplaza con una Cancelar botón y el Barra de progreso indica el progreso. Cuando esté terminado, verá un conjunto de Mandelbrot color:



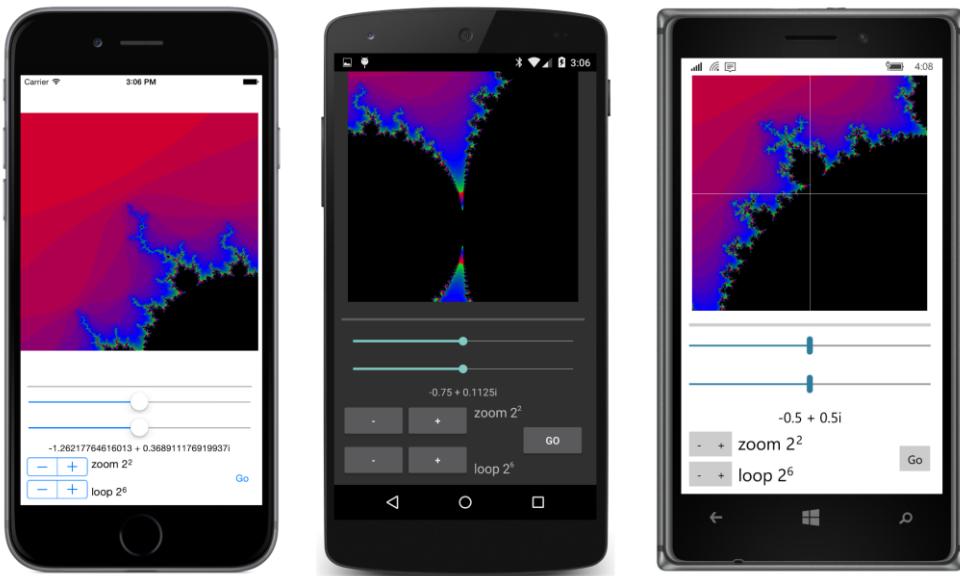
Se finaliza rápidamente debido a que el número máximo de iteraciones (indicado por la parte inferior paso a paso etiquetado lazo) está a sólo 2 a la tercera potencia, u 8. En consecuencia, el esquema del conjunto de Mandelbrot negro no es tan agudo como los programas anteriores. Muchos más puntos se marcan como ser un miembro del conjunto de lo que sería con un número de iteraciones máxima más alto. Puede aumentar el número de iteraciones que por potencias de 2. Aquí está una imagen más nítida con un número máximo de iteración de 64:



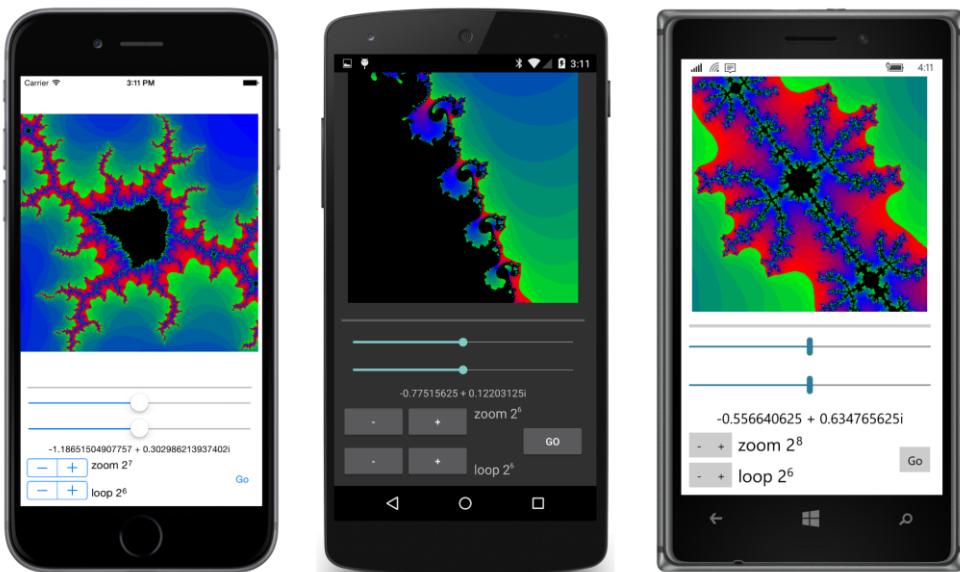
Los dos deslizadores vistas le permiten seleccionar un nuevo centro, que se muestra como un número complejo justo debajo de las barras de desplazamiento. El primer paso a paso elemento (etiquetado **enfocar**) le permite seleccionar un factor de magnificación, también en potencias de 2. Como se manipulan estos tres elementos, verá una caja con punto de mira construidos con seis delgada BoxView elementos. Esa caja marca el área que será magnificado la próxima vez que se pulsa el Ir botón:



Ahora pulse el Ir botón de nuevo y esperar. Ahora que el área previamente en caja llena el mapa de bits:



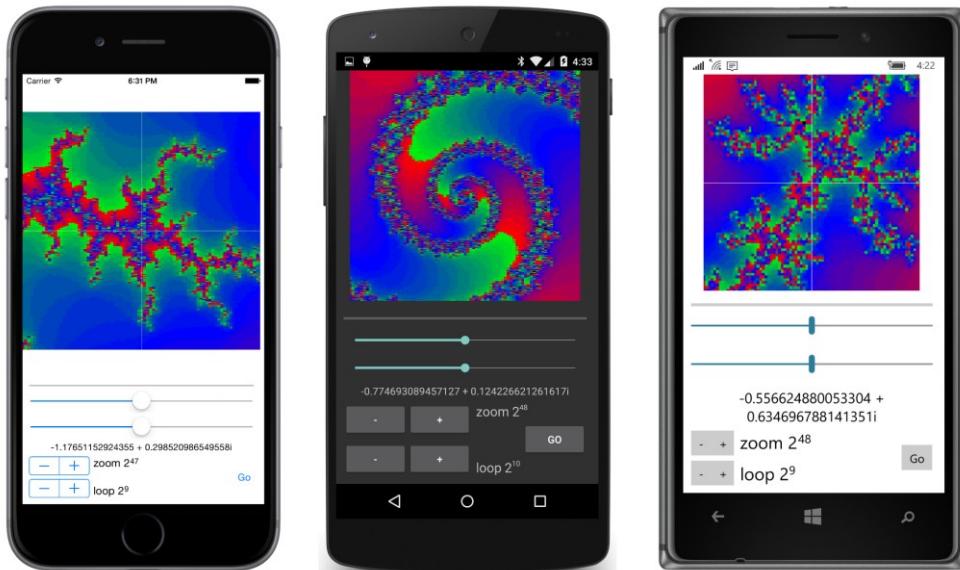
Después se calcula la nueva imagen, el punto de mira se recentrada, y se puede cambiar la posición del centro y un zoom de nuevo, y otra vez, y otra vez.



Sin embargo, en general, cuanto más se acerca la imagen, mayores serán las iteraciones máximas que necesitará para ver todos los detalles. Para cada dispositivo, la imagen en las capturas de pantalla anteriores adquiere visiblemente más detalle con cuatro veces más iteraciones:



Es una característica del conjunto de Mandelbrot que sólo puede mantener a acercar tanto como usted quiere y usted todavía ver apenas tanto detalle. Sin embargo, por lo general tendrá que seguir aumentando el recuento de iteración máxima, así, y para el momento de llegar a un factor de ampliación de 2 a la cuadragésima octava potencia más o menos, que ha alcanzado un techo que implica la resolución de flotante de doble precisión -Punto números. píxeles adyacentes ya no están asociados con números complejos distintos, y la imagen comienza a buscar bloques:



Eso no es un obstáculo fácil de superar. Existen implementaciones del números de punto flotante de precisión variable, pero debido a que no son manejados directamente por coprocesador matemático de la computadora, los cálculos relativos a estos números son necesariamente mucho más lento que flotador o doble tipos, y es probable que usted no va a querer el cálculo de Mandelbrot para ir más despacio.

los **MandelbrotXF** programa tiene tanto un modelo de vista y un modelo subyacente. El modelo hace el número real crujido y devuelve un objeto de tipo **BITMAPINFO**, lo que indica un ancho de pixel y la altura y una matriz de enteros. El tamaño de la matriz de enteros es el producto de la anchura de pixel y la altura, y los elementos de la matriz son de iteración recuentos. Un valor de -1 indica un miembro del conjunto de Mandelbrot:

```
espacio de nombres MandelbrotXF
{
    clase BITMAPINFO
    {
        público BITMAPINFO ( En t PixelWidth, En t pixelHeight, En t [] IterationCounts)
        {
            PixelWidth = PixelWidth;
            PixelHeight = pixelHeight;
            IterationCounts = iterationCounts;
        }

        public int PixelWidth { conjunto privado ; obtener ; }

        public int PixelHeight { conjunto privado ; obtener ; }

        public int [] IterationCounts { conjunto privado ; obtener ; }
    }
}
```

los MandelbrotModel clase contiene un único método asíncrono. Aparte de la IProgress

objeto, todos los argumentos son los tipos de valor, por lo que no hay peligro de cualquier argumento cambiando mientras que el cálculo está en curso:

```
espacio de nombres MandelbrotXF
{
    clase MandelbrotModel
    {
        público Tarea < BITMAPINFO > CalculateAsync ( Complejo Centrar,
                                                       doble anchura, doble altura,
                                                       En t PixelWidth, En t pixelHeight,
                                                       En t iteraciones,
                                                       IProgress < doble > Progreso,
                                                       CancellationToken cancelToken)

        {
            regreso Tarea .Llevar () =>
            {
                En t [] IterationCounts = new int [PixelWidth * pixelHeight];
                En t index = 0;

                para ( En t fila = 0; fila < pixelHeight; fila ++ )
                {

```

```

Informe de progreso(( doble ) De fila / pixelHeight);
cancelToken.ThrowIfCancellationRequested ();

doble y = Center.Imaginary - altura / 2 + fila * Altura / pixelHeight;

para ( En t col = 0; Col <PixelWidth; col++)
{
    doble x = Center.Real - ancho / 2 + col * ancho / PixelWidth;
    Complejo c = nuevo Complejo (X, y);

    Si ((C - nuevo Complejo (-1, 0)). MagnitudeSquared <1,0 / 16)
    {
        iterationCounts [índice ++] = -1;
    }
    // http://www.reenigne.org/blog/algoritm-for-mandelbrot-cardioid/
    else if (C.MagnitudeSquared * (8 * c.MagnitudeSquared - 3) <
            3.0 / 32 - C.Real)
    {
        iterationCounts [índice ++] = -1;
    }
    más
    {
        Complejo z = 0;
        En t iteración = 0;

        hacer
        {
            z = z * z + c;
            iteración++;
        }
        mientras (iteración <iteraciones && z.MagnitudeSquared <4);

        Si (iteración == iteraciones)
        {
            iterationCounts [índice ++] = -1;
        }
        más
        {
            iterationCounts [índice ++] = iteración;
        }
    }
}
return new BITMAPINFO (PixelWidth, pixelHeight, iterationCounts);
}, CancelToken);
}
}
}

Esta CalculateAsync método se llama solamente desde el modelo de vista. El modelo de vista también está destinado a proporcionar fuentes de enlace de datos para el archivo XAML y para ayudar al archivo de código subyacente en la realización de esos trabajos que los enlaces de datos XAML no pueden manejar. (Dibujo el punto de mira y cuadro de ampliación es un trabajo para ese archivo de código subyacente.)
```

Por esta razón, el MandelbrotViewModel clase tiene muchas propiedades, pero probablemente no las mismas propiedades que definiría si no estaba pensando en la interfaz de usuario. Los CurrentCenter propiedad es el número complejo para el centro de la imagen visualizada actualmente por el programa, y el CurrentMagnification también se aplica a la imagen. Pero el TargetMagnification está ligado a la configuración actual de la paso a paso, que se aplicará a la imagen siguiente calculado. Los RealOffset y Estoy-imaginarioOffset propiedades están obligados a los dos deslizador elementos y pueden ir de 0 a 1. En el CurrentCenter, CurrentMagnification, RealOffset, y ImaginaryOffset propiedades, el modelo de vista puede calcular el Target Center propiedad. Este es el centro de la imagen siguiente calculado. Como verá, que Target Center propiedad se utiliza para mostrar el número complejo por debajo de los dos deslizadores:

```
espacio de nombres MandelbrotXF
{
    clase MandelbrotViewModel : ViewModelBase
    {
        // Establecer a través de argumentos de constructor.
        sólo lectura doble baseWidth;
        sólo lectura doble baseHeight;

        // campos de apoyo para las propiedades.
        Complejo currentCenter, Target Center;
        Ent PixelWidth, pixelHeight;
        doble currentMagnification, targetMagnification;
        Ent iteraciones;
        doble realOffset, imaginaryOffset;
        bool está ocupado;
        doble Progreso;
        BITMAPINFO BITMAPINFO;

        público MandelbrotViewModel ( doble baseWidth, doble baseHeight)
        {
            esta .baseWidth = baseWidth;
            esta .baseHeight = baseHeight;

            // Crear un objeto MandelbrotModel.
            MandelbrotModel modelo = nuevo MandelbrotModel ();

            // reportero Progreso
            Progreso < doble > = ProgressReporter nuevo Progreso < doble > (( doble curso) =>
            {
                Progreso = progreso;
            });

            CancellationTokenSource cancelTokenSource = nulo ;

            // define CalculateCommand y CancelCommand.
            CalculateCommand = nuevo Mando (
                ejecutar: asíncrono () =>
                {
                    // Deshabilitar este botón y activar el botón Cancelar.
                    IsBusy = cierto ;
                    (( Mando ) CalculateCommand) .ChangeCanExecute ();
                }
            );
        }
    }
}
```

```
(( Mando ) CancelCommand) .ChangeCanExecute ();

// crear CancellationToken.
cancelTokenSource = nuevo CancellationTokenSource ();
CancellationToken cancelToken = cancelTokenSource.Token;

tratar
{
    // Realizar el cálculo.
    BIIMAPINFO = esperar model.CalculateAsync (Target Center,
                                                baseWidth / TargetMagnification,
                                                baseHeight / TargetMagnification,
                                                PixelWidth, PixelHeight,
                                                iteraciones,
                                                progressReporter,
                                                cancelToken);

    // Procesamiento sólo para una conclusión exitosa.
    CurrentCenter = Target Center;
    CurrentMagnification = TargetMagnification;
    RealOffset = 0,5;
    ImaginaryOffset = 0,5;
}

captura ( OperationCanceledException )
{
    // Operación cancelada!
}

captura
{
    // Otro tipo de excepción? Esto no debería ocurrir.
}

// Procesamiento independientemente del éxito o cancelación.
Progreso = 0;
IsBusy = falso ;

// Desactivar botón Cancelar y activar este botón.
(( Mando ) CalculateCommand) .ChangeCanExecute ();
(( Mando ) CancelCommand) .ChangeCanExecute ();
},
CanExecute: () =>
{
    regreso !Está ocupado;
});

CancelCommand = nuevo Mando (
    ejecutar: () =>
    {
        cancelTokenSource.Cancel ();
    },
    CanExecute: () =>
    {
        regreso Está ocupado;
    });
}
```

```

    }

    public int PixelWidth
    {
        conjunto { SetProperty ( árbito PixelWidth, valor ); }
        obtener { regreso PixelWidth; }
    }

    public int PixelHeight
    {
        conjunto { SetProperty ( árbito pixelHeight, valor ); }
        obtener { regreso pixelHeight; }
    }

    público Complejo CurrentCenter
    {
        conjunto
        {
            Si ( SetProperty ( árbito currentCenter, valor ))
                CalculateTargetCenter ();
            }
            obtener { regreso currentCenter; }
        }
    }

    público Complejo Target Center
    {
        conjunto privado { SetProperty ( árbito Target Center, valor ); }
        obtener { regreso Target Center; }
    }

    pública doble CurrentMagnification
    {
        conjunto { SetProperty ( árbito currentMagnification, valor ); }
        obtener { regreso currentMagnification; }
    }

    pública doble TargetMagnification
    {
        conjunto { SetProperty ( árbito targetMagnification, valor ); }
        obtener { regreso targetMagnification; }
    }

    public int iteraciones
    {
        conjunto { SetProperty ( árbito iteraciones, valor ); }
        obtener { regreso iteraciones; }
    }

    // Estas dos propiedades varían de 0 a 1.
    // Indican un nuevo centro en relación con el
    // anchura actual y la altura, que es el baseWidth
    // y baseHeight dividido por CurrentMagnification.
    pública doble RealOffset
    {

```

```

conjunto
{
    Si ( SetProperty ( árbito realOffset, valor ))
        CalculateTargetCenter ();
    }
    obtener { regreso realOffset; }
}

pública doble ImaginaryOffset
{
    conjunto
    {
        Si ( SetProperty ( árbito imaginaryOffset, valor ))
            CalculateTargetCenter ();
        }
        obtener { regreso imaginaryOffset; }
    }

vacio CalculateTargetCenter ()
{
    doble width = baseWidth / CurrentMagnification;
    doble height = baseHeight / CurrentMagnification;

    Target Center = nuevo Complejo ( CurrentCenter.Real + (RealOffset - 0,5) * anchura,
                                    CurrentCenter.Imaginary + (ImaginaryOffset - 0,5) *
                                    altura);
}

public bool Está ocupado
{
    conjunto privado { SetProperty ( árbito está ocupado, valor ); }
    obtener { regreso está ocupado; }
}

pública doble Progreso
{
    conjunto privado { SetProperty ( árbito Progreso, valor ); }
    obtener { regreso Progreso; }
}

público BITMAPINFO BITMAPINFO
{
    conjunto privado { SetProperty ( árbito BITMAPINFO, valor ); }
    obtener { regreso BITMAPINFO; }
}

público Yo ordeno CalculateCommand { conjunto privado ; obtener ; }

público Yo ordeno CancelCommand { conjunto privado ; obtener ; }
}
}

```

MandelbrotViewModel también define dos propiedades de tipo Yo ordeno Para el Calcular y Cancelar botones, una Progreso propiedad, y una Está ocupado propiedad. Como verá, la Está ocupado propiedad se utiliza

para mostrar uno de los dos botones y ocultar la otra y para desactivar el resto de la interfaz de usuario durante los cálculos. Los dos **Yo ordeno propiedades** se implementan con las funciones lambda en el constructor de la clase.

Los enlaces de datos en el archivo XAML a las propiedades en MandelbrotViewModel requerir dos nuevos convertidores de obligado cumplimiento del **Xamarin.FormsBook.Toolkit** biblioteca. La primera simplemente niega una bool valor:

```
espacio de nombres Xamarin.FormsBook.Toolkit
{
    clase pública BooleanNegationConverter : IValueConverter
    {
        objeto público Convertir( objeto valor, Tipo tipo de objetivo,
                                  objeto parámetro, CultureInfo cultura)
        {
            regreso !( bool )valor;
        }

        objeto público ConvertBack ( objeto valor, Tipo tipo de objetivo,
                                    objeto parámetro, CultureInfo cultura)
        {
            regreso !( bool )valor;
        }
    }
}
```

Esto se utiliza en conjunción con el **Está ocupado** propiedad del modelo de vista. Cuando **Está ocupado** es cierto, el **Está habilitado** propiedades de varios elementos y la **Es visible** propiedad de la **Ir** botón es necesario establecer a falso.

Ambos paso a paso elementos controlan realmente un exponente de un valor en el modelo de vista. UN paso a paso valor de 8, por ejemplo, corresponde a una iteraciones o TargetMagnification valor de 256. Esta conversión requiere un convertidor de logaritmo en base 2:

```
espacio de nombres Xamarin.FormsBook.Toolkit
{
    clase pública BaseTwoLogConverter : IValueConverter
    {
        objeto público Convertir( objeto valor, Tipo tipo de objetivo,
                                  objeto parámetro, CultureInfo cultura)
        {
            Si ( valor es int )
            {
                regreso Mates .Iniciar sesión(( Ent ) Valor) / Mates .Log (2);
            }
            regreso Mates .Iniciar sesión(( doble ) Valor) / Mates .Log (2);
        }

        objeto público ConvertBack ( objeto valor, Tipo tipo de objetivo,
                                    objeto parámetro, CultureInfo cultura)
        {
            doble returnValue = Mates .Pow (2, ( doble )valor);

            Si (TargetType == tipo de ( Ent ))

```

```
    {
        regreso ( En t ) ReturnValue;
    }
    regreso ReturnValue;
```

Aquí está el archivo XAML, con fijaciones a la Progreso, RealOffset, ImaginaryOffset, TargetCenter, TargetMagnification, Iteraciones, IsBusy, CalculateCommand, y CancelCommand propiedades del ViewModel:

```

<AbsoluteLayout x: Nombre = "crossHairLayout"
    Grid.Row = "0" Grid.Column = "0"
    HorizontalOptions = "Centrar"
    VerticalOptions = "Centrar"
    SizeChanged = "OnCrossHairLayoutSizeChanged" >

    <AbsoluteLayout.Resources>
        <ResourceDictionary>
            <Estilo Tipo de objetivo = "BoxView" >
                <Setter Propiedad = "Color" Valor = "Blanco" />
                <Setter Propiedad = "AbsoluteLayout.LayoutBounds" Valor = "0,0,0,0" />
            </Estilo>
        </ResourceDictionary>
    </AbsoluteLayout.Resources>

    <BoxView x: Nombre = "realCrossHair" />
    <BoxView x: Nombre = "imagCrossHair" />
    <BoxView x: Nombre = "caja superior" />
    <BoxView x: Nombre = "bottomBox" />
    <BoxView x: Nombre = "leftbox" />
    <BoxView x: Nombre = "rightbox" />
</AbsoluteLayout>

<StackLayout x: Nombre = "controlPanelStack"
    Grid.Row = "1" Grid.Column = "0"
    Relleno = "10" >

    <Barra de progreso Progreso = "0 La unión Progreso" >
        VerticalOptions = "CenterAndExpand" />

    <StackLayout VerticalOptions = "CenterAndExpand" >
        <deslizador Valor = "(Binding RealOffset, Mode = TwoWay)" >
            Está habilitado = "(Binding IsBusy, convertidor = {}) negate StaticResource" />
        <deslizador Valor = "(Binding ImaginaryOffset, Mode = TwoWay)" >
            Está habilitado = "(Binding IsBusy, convertidor = {}) negate StaticResource" />
        <Etiqueta Texto = "{(Binding Target Center, StringFormat = '{0})}" >
            Tamaño de fuente = "Pequeña"
            HorizontalTextAlignment = "Centrar" />
    </StackLayout>

    <Cuadrícula VerticalOptions = "CenterAndExpand" >
        <Grid.RowDefinitions>
            <RowDefinition Altura = "Auto" />
            <RowDefinition Altura = "Auto" />
        </Grid.RowDefinitions>

        <Grid.ColumnDefinitions>
            <ColumnDefinition Anchura = "Auto" />
            <ColumnDefinition Anchura = "*" />
        </Grid.ColumnDefinitions>

```

```

<!-- factor de aumento paso a paso y la pantalla. -->
< paso a paso x: Nombre = " magnificationStepper "
  Grid.Row = " 0 " Grid.Column = " 0 "
  Valor = " {Binding TargetMagnification,
    Convertidor = ()} base2log StaticResource "
  Está habilitado = " {Binding IsBusy, convertidor = ()} negate StaticResource "
  VerticalOptions = " Centrar " />

< StackLayout Grid.Row = " 0 " Grid.Column = " 1 "
  Orientación = " Horizontal "
  Espaciado = " 0 "
  VerticalOptions = " comienzo " >
  < Etiqueta Texto = " zoom 2 "
    Tamaño de fuente = " Medio " />
  < Etiqueta Texto = " {Binding Fuente = {x: magnificationStepper Referencia},
    Path = Valor,
    StringFormat = '{0}' } "
    Tamaño de fuente = " Micro " />
</ StackLayout >

<!-- Iteraciones factor paso a paso y la pantalla. -->
< paso a paso x: Nombre = " iterationsStepper "
  Grid.Row = " 1 " Grid.Column = " 0 "
  Valor = " {Iteraciones Encuadernación, convertidor = ()} StaticResource base2log "
  Está habilitado = " {Binding IsBusy, convertidor = ()} negate StaticResource "
  VerticalOptions = " Centrar " />

< StackLayout Grid.Row = " 1 " Grid.Column = " 1 "
  Orientación = " Horizontal "
  Espaciado = " 0 "
  VerticalOptions = " Fin " >
  < Etiqueta Texto = " bucle 2 "
    Tamaño de fuente = " Medio " />
  < Etiqueta Texto = " {Binding Fuente = {x: iterationsStepper Referencia},
    Path = Valor,
    StringFormat = '{0}' } "
    Tamaño de fuente = " Micro " />
</ StackLayout >

<!-- GO / Cancelar. -->
< Cuadrícula Grid.Row = " 0 " Grid.Column = " 1 " Grid.RowSpan = " 2 "
  HorizontalOptions = " Fin "
  VerticalOptions = " Centrar " >

  < Botón Texto = " Ir "
    Mando = " {} La unión CalculateCommand "
    Es visible = " {Binding IsBusy, convertidor = ()} negate StaticResource " />

  < Botón Texto = " Cancelar "
    Mando = " {} La unión CancelCommand "
    Es visible = " {} La unión IsBusy " />
</ Cuadrícula >
</ Cuadrícula >
</ StackLayout >

```

[« Cuadrícula »](#)
[« Página de contenido »](#)

Este archivo XAML sólo instala tres controladores de eventos, y todos ellos son SizeChanged manipuladores.

El primero SizeChanged manejador es en la propia página. Este controlador es utilizado por el archivo de código subyacente para adaptarse mainGrid y sus hijos para las técnicas de modo horizontal o vertical utilizando ha visto en las muestras anteriores.

El segundo SizeChanged controlador está en el Imagen elemento. El archivo de código subyacente utiliza esto para el tamaño AbsoluteLayout que muestra el punto de mira y cuadro de ampliación. Esta AbsoluteLayout se debe hacer en el mismo tamaño que el mapa de bits que muestra el Imagen bajo la suposición de que la Imagen se mostrará un mapa de bits cuadrado.

El tercero SizeChanged manejador está en ese AbsoluteLayout, así el punto de mira y cuadro de ampliación pueden ser rediseñados para un cambio en el tamaño.

los MandelbrotXF programa también realiza un pequeño truco de clases para asegurar que el mapa de bits contiene el número óptimo de píxeles, lo que sucede cuando hay una correspondencia de uno a uno entre los píxeles del mapa de bits y los píxeles de la pantalla. El archivo XAML contiene un segundo Imagen elemento llamado

TestImage. Esta Imagen es invisible debido a que la Opacidad se ajusta a cero, y se horizontal y verticalmente centrada, lo que significa que se visualizará con un mapeo de píxeles uno-a-uno. El archivo de código subyacente crea un mapa de bits cuadrada 120 píxeles que se establece en este Imagen. El tamaño resultante de la Imagen permite que el programa sabe el número de píxeles que hay en la unidad independiente del dispositivo, y se puede utilizar para calcular que un tamaño óptimo de píxeles para el mapa de bits de Mandelbrot. (Por desgracia, no funciona para las plataformas de Windows en tiempo de ejecución.)

Aquí es más o menos la primera mitad del archivo de código subyacente para MandelbrotXFPage, mostrando sobre todo la creación de instancias de la MandelbrotViewModel clase y la interacción de estos SizeChanged manipuladores:

```
espacio de nombres MandelbrotXF
{
    público clase parcial MandelbrotXFPage : Pagina de contenido
    {
        MandelbrotViewModel mandelbrotViewModel;
        doble pixelsPerUnit = 1;

        público MandelbrotXFPage ()
        {
            InitializeComponent ();

            // Instantiate modelo de vista y obtener los valores guardados.
            mandelbrotViewModel = nuevo MandelbrotViewModel (2.5, 2.5)
            {
                PixelWidth = 1,000,
                PixelHeight = 1,000,
                CurrentCenter = nuevo Complejo (GetProperty ("CenterReal", -0,75),
                                                GetProperty ("CenterImaginary", 0,0)),
                CurrentMagnification = GetProperty ("Aumento", 1,0),
                TargetMagnification = GetProperty ("Aumento", 1,0),
            }
        }
    }
}
```

```
Iteraciones = GetProperty ( "iteraciones" , 8),
RealOffset = 0,5,
ImaginaryOffset = 0,5
};

// Conjunto BindingContext en la página.
BindingContext = mandelbrotViewModel;

// manipulador Conjunto PropertyChanged el modelo de vista para su procesamiento "manual".
mandelbrotViewModel.PropertyChanged += OnMandelbrotViewModelPropertyChanged;

// Crear una imagen de prueba para obtener píxeles por unidad independiente del dispositivo.
BmpMaker bmpMaker = nuevo BmpMaker (120, 120);

testImage.SizeChanged += (remitente, args) =>
{
    pixelsPerUnit = bmpMaker.Width / testImage.Width;
    SetPixelWidthAndHeight ();
};

testImage.Source = bmpMaker.Generate ();

// reducir gradualmente la opacidad del punto de mira.
Dispositivo .StartTimer ( Espacio de tiempo .FromMilliseconds (100), () =>
{
    realCrossHair.Opacity - = 0.01;
    imagCrossHair.Opacity - = 0.01;
    return true ;
});

// método para acceder a Propiedades del diccionario si la clave no está todavía presente.
T GetProperty <T> ( cuerda clave, T defaultValue)
{
    IDictionary < cuerda , objeto > properties = Solicitud .Current.Properties;

    Si (Properties.ContainsKey (clave))
    {
        regreso (T) propiedades [tecla];
    }
    regreso valor por defecto;
}

// Cambiar entre modo vertical y horizontal.
vacío OnPageSizeChanged ( objeto remitente, EventArgs args)
{
    Si (== Anchura -1 || Altura == -1)
        regreso ;

    // Modo retrato.
    Si (Anchura <Altura)
    {
        mainGrid.RowDefinitions [1] .height = GridLength .Auto;
        mainGrid.ColumnDefinitions [1] .Width = nuevo GridLength (0, GridUnitType .Absoluto);
        Cuadricula .SetRow (controlPanelStack, 1);
    }
}
```

```

        Cuadrícula .SetColumn (controlPanelStack, 0);
    }
    // Modo paisaje.
    más
    {
        mainGrid.RowDefinitions [1] .height = nuevo GridLength (0, GridUnitType .Absoluto);
        mainGrid.ColumnDefinitions [1] .Width = nuevo GridLength (1, GridUnitType .Estrella);
        Cuadrícula .SetRow (controlPanelStack, 0);
        Cuadrícula .SetColumn (controlPanelStack, 1);
    }
}

vacío OnImageSizeChanged ( objeto remitente, EventArgs args)
{
    // Asegurarse de que la disposición de cruz se encuentre mismo tamaño que la imagen.
    doble size = Mates .min (image.Width, image.Height);
    crossHairLayout.WidthRequest = tamaño;
    crossHairLayout.HeightRequest = tamaño;

    // Calcular el tamaño en pixeles del elemento de imagen.
    SetPixelWidthAndHeight ();
}

// Establece el mapa de bits de Mandelbrot para ancho de píxel óptima y la altura.
vacío SetPixelWidthAndHeight ()
{
    En t pixeles = ( En t ) (* PixelsPerUnit Mates .min (image.Width, image.Height));
    mandelbrotViewModel.PixelWidth = pixeles;
    mandelbrotViewModel.PixelHeight = pixeles;
}

// Volver a dibujar mira si el diseño en forma de cruz cambia de tamaño.
vacío OnCrossHairLayoutSizeChanged ( objeto remitente, EventArgs args)
{
    SetCrossHairs ();
}

...
}

}

```

En lugar de unir un grupo de controladores de eventos a los elementos de interfaz de usuario en el archivo XAML, el constructor del archivo de código subyacente en vez concede una **PropertyChanged manejador a la Mandelbrot-**

ViewModel ejemplo. Los cambios en varias propiedades requieren que el punto de mira y caja de medición volver a dibujar, y cualquier cambio a cualquier propiedad aporta el punto de mira de nuevo en vista:

```

espacio de nombres MandelbrotXF
{
    ...
    vacío asíncrono OnMandelbrotViewModelPropertyChanged ( objeto remitente,
        PropertyChangedEventArgs args)

```

```
{  
    // Establecer la opacidad de nuevo a 1.  
    realCrossHair.Opacity = 1;  
    imagCrossHair.Opacity = 1;  
  
    cambiar (Args.PropertyName)  
    {  
        caso "RealOffset":  
        caso "ImaginaryOffset":  
        caso "CurrentMagnification":  
        caso "TargetMagnification":  
            // Volver a dibujar mira si estas propiedades cambian  
            SetCrossHairs ();  
            descanso ;  
  
        caso "BITMAPINFO":  
            // crear mapa de bits basado en el recuento de iteración.  
            DisplayNewBitmap (mandelbrotViewModel.BitmapInfo);  
  
            // guardar propiedades para el próximo programa vez que se ejecute.  
            IDictionary < cuerda , objeto > properties = Solicitud .Current.Properties;  
            propiedades [ "CenterReal" ] = MandelbrotViewModel.TargetCenter.Real;  
            propiedades [ "CenterImaginary" ] = MandelbrotViewModel.TargetCenter.Imaginary;  
            propiedades [ "Aumento" ] = MandelbrotViewModel.TargetMagnification;  
            propiedades [ "iteraciones" ] = mandelbrotViewModel.Iterations;  
            esperar Solicitud .Current.SavePropertiesAsync ();  
            descanso ;  
    }  
}  
  
vacio SetCrossHairs ()  
{  
    // tamaño del diseño para el punto de mira y el zoom de cuadro.  
    tamaño layoutSize = nuevo tamaño (CrossHairLayout.Width, crossHairLayout.Height);  
  
    // posición fraccionada del centro de punto de mira.  
    doble xcenter = mandelbrotViewModel.RealOffset;  
    doble ycenter = 1 - mandelbrotViewModel.ImaginaryOffset;  
  
    // Calcular la dimensión del cuadro de zoom.  
    doble boxSize = mandelbrotViewModel.CurrentMagnification /  
        mandelbrotViewModel.TargetMagnification;  
  
    // posiciones fraccionarias de las esquinas de la caja de zoom.  
    doble xLeft = xcenter - boxSize / 2;  
    doble xLeft = xcenter + boxSize / 2;  
    doble yTop = ycenter - boxSize / 2;  
    doble yBottom = ycenter + boxSize / 2;  
  
    // Establecer todos los límites de diseño.  
    SetLayoutBounds (realCrossHair,  
        nuevo Rectángulo (xcenter, yTop, 0, boxSize),  
        layoutSize);  
    SetLayoutBounds (imagCrossHair,
```

```

        nuevo Rectángulo (xLeft, ycenter, boxSize, 0),
        layoutSize);
    SetLayoutBounds (topbox, nuevo Rectángulo (xLeft, yTop, boxSize, 0), layoutSize);
    SetLayoutBounds (bottomBox, nuevo Rectángulo (xLeft, yBottom, boxSize, 0), layoutSize);
    SetLayoutBounds (leftbox, nuevo Rectángulo (xLeft, yTop, 0, boxSize), layoutSize);
    SetLayoutBounds (rightbox, nuevo Rectángulo (xLeft, yTop, 0, boxSize), layoutSize);
}

vacio (SetLayoutBounds Ver ver, Rectángulo fractionalRect, tamaño layoutSize)
{
    Si (LayoutSize.Width == -1 || layoutSize.Height == -1)
    {
        AbsoluteLayout .SetLayoutBounds (vista, nuevo Rectángulo ());
        regreso ;
    }

    const doble espesor = 1;
    Rectángulo absoluteRect = nuevo Rectángulo ();

    // Lineas horizontales.
    Si (FractionalRect.Height == 0 && fractionalRect.Y > 0 && fractionalRect.Y <1)
    {
        doble xLeft = Mates .MAX (0, fractionalRect.Left);
        doble xLeft = Mates .min (1, fractionalRect.Right);
        absoluteRect = nuevo Rectángulo (* LayoutSize.Width xLeft,
                                         layoutSize.Height * fractionalRect.Y,
                                         layoutSize.Width * (xLeft - xLeft),
                                         espesor);
    }
    // Lineas verticales.
    else if (FractionalRect.Width == 0 && fractionalRect.X > 0 && fractionalRect.X <1)
    {
        doble yTop = Mates .MAX (0, fractionalRect.Top);
        doble yBottom = Mates .min (1, fractionalRect.Bottom);
        absoluteRect = nuevo Rectángulo (* LayoutSize.Width fractionalRect.X,
                                         layoutSize.Height * yTop,
                                         espesor,
                                         layoutSize.Height * (yBottom - yTop));
    }
    AbsoluteLayout .SetLayoutBounds (vista, absoluteRect);
}
...
}
}

```

Las primeras versiones del programa intentaron utilizar la facilidad de dimensionamiento y posicionamiento proporcional de ABSoluteLayout para los seis BoxView elementos, pero llegaron a ser demasiado difícil. Los valores fraccionarios se pasan a la SetLayoutBounds método, pero los que se utilizan para calcular las coordenadas en función del tamaño de la AbsoluteLayout.

Dado que los modelos y ViewModels se supone que son independientes de la plataforma, ni Mandelbrot-Modelo ni MandelbrotViewModel involucrarse con la creación del mapa de bits real. Estas clases expresan la imagen como una BITMAPINFO valor, que es simplemente un ancho de píxel y la altura y una matriz de enteros que corresponden a iteración recuentos. La creación y visualización de mapa de bits que implica el uso de su mayoría bmp-

Fabricante y la aplicación de un esquema de color basado en el número de iteraciones:

```

espacio de nombres MandelbrotXF
{
    ...
    vacío DisplayNewBitmap (BITMAPINFO BITMAPINFO)
    {
        // crear el mapa de bits.
        BmpMaker bmpMaker = nuevo BmpMaker (BitmapInfo.PixelWidth, bitmapInfo.PixelHeight);

        // Establecer los colores.
        En t index = 0;
        para (En t fila = 0; fila <bitmapInfo.PixelHeight; fila++)
        {
            para (En t col = 0; Col <bitmapInfo.PixelWidth; col++)
            {
                En t iterationCount = bitmapInfo.IterationCounts [índice ++];

                // En el conjunto de Mandelbrot: negro color.
                Si (IterationCount == -1)
                {
                    bmpMaker.SetPixel (fila, col, 0, 0, 0);
                }
                // No en el conjunto de Mandelbrot: Elija un color basado en el recuento.
                más
                {
                    doble proporción = (iterationCount / 32,0)% 1;

                    Si (Proporción <0,5)
                    {
                        bmpMaker.SetPixel (fila, col, (En t) (255 * (1 - 2 * proporción)),
                            0,
                            (En t) (255 * 2 * proporción));
                    }
                    más
                    {
                        proporción = 2 * (proporción - 0,5);
                        bmpMaker.SetPixel (fila, col, 0,
                            (En t) (255 * proporción),
                            (En t) (255 * (1 - proporción)));
                    }
                }
            }
        }
        image.Source = bmpMaker.Generate ();
    }
}

```

Siéntase libre de experimentar con la combinación de colores. Una alternativa fácil es variar la tonalidad de un color HSL con el número de iteraciones:

```
double hue = (iterationCount / 64,0)% 1;
bmpMaker.SetPixel (fila, col, Color.FromHsla (matiz, 1, 0,5));
```

Volver a la web

Antes de este capítulo, el único código asíncrono en este libro que participan web accede utilizando la única clase razonable disponible para ese propósito en la biblioteca de clases portátil, `WebRequest`. Los `WebRequest`

clase utiliza un protocolo asíncrono mayor llamada el modelo de programación asíncrona o APM. APM implica dos métodos, en el caso de `WebRequest`, estos son llamados `BeginGetResponse` y `EndGetResponse`.

Puede convertir este par de llamadas de método en el modelo asíncrono basado en tareas (TAP) mediante el uso de la `FromAsync` método de `TaskFactory`, y el `ApmToTap` programa demuestra cómo. El programa utiliza un acceso a la Web y `ImageSource.FromStream` para cargar un mapa de bits y mostrarlo. Esta técnica se muestra en el Capítulo 13 como una alternativa a `ImageSource.FromUri`.

El archivo contiene un XAML Imagen elemento de espera de un mapa de bits, una ActivityIndicator que se ejecuta cuando el mapa de bits se está cargando, una Etiqueta para mostrar un posible mensaje de error, y una Botón para iniciar la descarga:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " ApmToTap.ApmToTapPage " >

    < ContentPage.Padding >
        < OnPlatform x: TypeArguments = " Espesor "
            iOS = " 0, 20, 0, 0 " >
    </ ContentPage.Padding >

    < StackLayout >
        < Cuadrícula VerticalOptions = " FillAndExpand " >
            < Etiqueta x: Nombre = " errorLabel "
                HorizontalOptions = " Centrar "
                VerticalOptions = " Centrar " >
                < ActivityIndicator Esta comiendo = " {Binding Fuente = {x: Referencia},
                    Path = IsLoading} " />
                < Imagen x: Nombre = " imagen " />
            </ Cuadrícula >
            < Botón Texto = " carga de mapa de bits "
                HorizontalOptions = " Centrar "
                hecho clic = " OnLoadButtonClicked " />
        </ StackLayout >
    </ Página de contenido >
```

El archivo de código subyacente consolida toda la WebRequest código en un método llamado asíncrono `GetStreamAsync`. Después de la `TaskFactory` y `WebRequest` objetos se instancian, el método pasa a la `BeginGetResponse` y `EndGetResponse` métodos a la `FromAsync` método de Tarea-fábrica, que luego devuelve una `WebResponse` objeto desde el que una Corriente está disponible:

```
público clase parcial ApmToTapPage : Pagina de contenido
{
    público ApmToTapPage ()
    {
        InitializeComponent ();
    }

    vacío asíncrono OnLoadButtonClicked ( objeto remitente, EventArgs args)
    {
        tratar
        {
            Corriente flujo =
                esperar GetStreamAsync ( "Https://developer.xamarin.com/demo/IMG_1996.JPG" );
            image.Source = Fuente de imagen .FromStream (( ) => corriente);
        }
        captura ( Excepción Exc*)
        {
            errorLabel.Text = exc.Message;
        }
    }

    esíncrono Tarea < Corriente > GetStreamAsync ( cuerda URI)
    {
        TaskFactory de fábrica = nuevo TaskFactory ();
        WebRequest request = WebRequest .Create (URI);
        WebResponse respuesta = esperar factory.FromAsync<WebResponse> (Request.BeginGetResponse,
            request.EndGetResponse,
            nulo );
        regreso response.GetResponseStream ();
    }
}
```

los hecho clic controlador para el Botón a continuación, puede obtener esa Corriente objeto llamando `GetStreamAsync` con un URI. Como de costumbre, el código con el `esperar` operador está en una `tratar` bloquear para detectar los errores posibles. Puede experimentar un poco por falta de ortografía deliberadamente el dominio o nombre de archivo para ver qué tipo de errores que se obtiene.

Otra opción para Web accede a una clase llamada `HttpClient` en el `System.Net.Http` espacio de nombres. Esta clase no está disponible en la versión de .NET incluido en la biblioteca de clases portátil en una solución Xamarin.Forms, pero Microsoft ha hecho la clase disponible como un paquete NuGet:

<https://www.nuget.org/packages/Microsoft.Net.Http>

Desde el gestor de NuGet en Visual Studio o Xamarin de estudio, sólo la búsqueda de "HttpClient".

`HttpClient` se basa en TAP. Los métodos asíncronos regresan Tarea y Tarea <T> objetos, y algunos de los métodos también tienen `CancellationToken` argumentos.

Ninguno de los avances informe métodos, sin embargo, lo que sugiere que una clase moderna de primer orden para web accede todavía no está disponible para bibliotecas de clases portátiles.

En el siguiente capítulo verá muchos más usos de esperar y explorar algunas otras características del modelo asíncrono basado en tareas en relación con la aplicación emocionantes Xamarin.Forms de la animación.

capítulo 21

transformadas

Con la ayuda de StackLayout y Cuadricula, Xamarin.Forms hace un buen trabajo de dimensionamiento y posicionamiento de los elementos visuales de la página. A veces, sin embargo, es necesario (o conveniente) para la aplicación que hacer algunos ajustes. Es posible que desee para compensar la posición de los elementos tanto, cambiar su tamaño, o incluso rotarlas.

son posibles Tales cambios en la ubicación, el tamaño, o la orientación usando una característica de Xamarin.Forms conocidos como *transforma*. El concepto de la transformada se originó en la geometría. La transformada es una fórmula que asigna puntos a otros puntos. Por ejemplo, si desea cambiar de un objeto geométrico en un sistema de coordenadas cartesianas, puede agregar factores de desplazamiento constante a todas las coordenadas que definen ese objeto.

Estas transformaciones geométricas, matemáticas juegan un papel vital en la programación de gráficos por ordenador, donde a veces se conocen como *transformaciones de matriz* porque son más fáciles de expresar matemáticamente usando el álgebra matricial. Sin transforma, no puede haber ningún tipo de gráficos en 3D. Pero con los años, las transformaciones han emigrado de programación gráfica para la programación de interfaz de usuario. Todas las plataformas soportadas por Xamarin.Forms apoyan transformadas básicas que se pueden aplicar a los elementos de interfaz de usuario tales como texto, mapas de bits, y botones.

Xamarin.Forms soporta tres tipos básicos de las transformaciones:

- *Traducción* -shiftin un elemento horizontal o vertical o ambos.
- *Escala* -cambiando el tamaño de un elemento.
- *Rotación* -rechazar un elemento alrededor de un punto o eje.

El escalado con el apoyo de Xamarin.Forms es uniforme en todas las direcciones, técnicamente conocida como *isotrópico* escalada. No se puede utilizar la escala para cambiar la relación de aspecto de un elemento visual. La rotación es compatible tanto con la superficie de dos dimensiones de la pantalla y en el espacio 3D. Xamarin.Forms no admite un sesgo transformar o transformar una matriz generalizada.

Xamarin.Forms apoya estas transformadas con ocho propiedades de la VisualElement clase. Estas propiedades son todas de tipo doble:

- TranslationX
- TranslationY
- Escala
- Rotación
- rotationX

- rotationY
- anchorX
- anchorY

Como se verá en el siguiente capítulo, Xamarin.Forms también tiene un sistema de animación extensa y extensible que puede apuntar estas propiedades. Pero también se puede realizar a transformar animaciones por su cuenta mediante el uso de `Device.StartTimer` o `Task.Delay`. En este capítulo se demuestra algunas técnicas de animación y tal vez le ayudará a entrar en un fotograma de la animación de la mente en la preparación para el Capítulo 22.

La traducción transformar

Una aplicación utiliza uno de la disposición classes- `StackLayout`, Cuadrícula, `AbsoluteLayout`, o Relativo-

Diseño -para posicionar un elemento visual en la pantalla. Vamos a llamar a la posición establecida por el sistema de diseño de la "posición de diseño."

los valores no nulos de la `TranslationX` y `TranslationY` propiedades cambian la posición de un elemento visual relativa a la posición de disposición. Los valores positivos de `TranslationX` desplazar el elemento hacia la derecha, y los valores positivos de la `TranslationY` desplazar el elemento hacia abajo.

los `TranslationDemo` programa le permite experimentar con estas dos propiedades. Todo está en el archivo XAML:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " TranslationDemo.TranslationDemoPage " >

    < StackLayout Relleno = " 20, 10 " >
        < Marco x: Nombre = " marco "
            HorizontalOptions = " Centrar "
            VerticalOptions = " CenterAndExpand "
            OutlineColor = " Acento " >

            < Etiqueta Texto = " TEXTO "
                Tamaño de fuente = " Grande " />
        </ Marco >

        < deslizador x: Nombre = " xSlider "
            Mínimo = " -200 "
            Máximo = " 200 "
            Valor = " {Binding Fuente = {x: Marco de Referencia},
                Path = TranslationX} " />

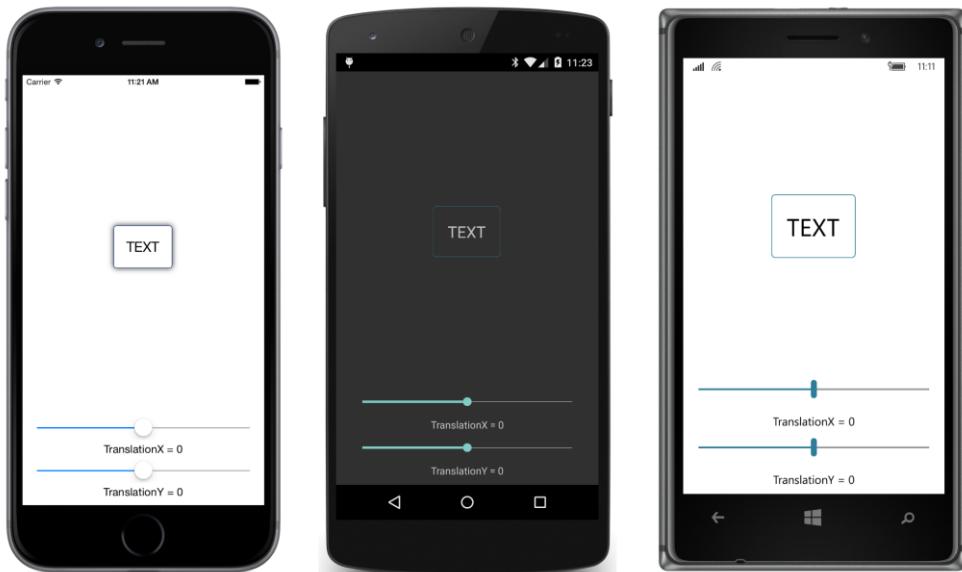
        < Etiqueta Texto = " {Binding Fuente = {x: xSlider Referencia},
                Path = Valor,
                StringFormat = " TranslationX = (0: F0) " }
            HorizontalTextAlignment = " Centrar " />

        < deslizador x: Nombre = " ySlider "
            Mínimo = " -200 "
            Máximo = " 200 "
            Valor = " {Binding Fuente = {x: ySlider Referencia},
                Path = Valor,
                StringFormat = " TranslationY = (0: F0) " } " />
    </ StackLayout >
```

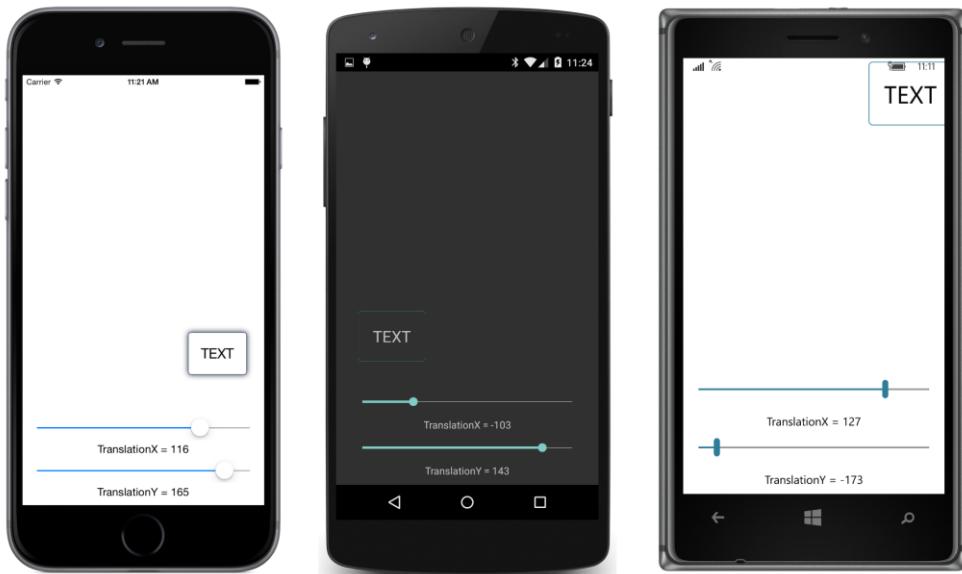
```
    Mínimo = "-200"
    Máximo = "200"
    Valor = "{Binding Fuente = {x: Marco de Referencia},
              Path = TranslationY}" />

    < Etiqueta Texto = "{Binding Fuente = {x: ySlider Referencia}},
      Path = Valor,
      StringFormat = 'TranslationY = {0: F0}' "
      HorizontalTextAlignment = "Centrar" />
  </ StackLayout >
</ Página de contenido >
```

UN Marco encierra una Etiqueta y está centrado en la parte superior de la StackLayout. Dos deslizador elementos tienen enlaces con el TranslationX y TranslationY propiedades de la Marco, y se inicializan para un rango de -200 a 200. La primera vez que ejecute el programa, los dos deslizadores se establecen en los valores por defecto de TranslationX y TranslationY, que son cero:



Puede manipular las barras de desplazamiento para mover el Marco alrededor de la pantalla. Los valores de TranslationX y TranslationY especificar un desplazamiento del elemento con respecto a su posición de diseño original:



Si los valores son lo suficientemente grandes, el elemento puede ser traducido a otros elementos visuales se superponen o se mueva fuera de la pantalla por completo.

Una traducción de un elemento tal como una Marco También afecta a todos los hijos de ese elemento, que en este caso es sólo el Etiqueta. Puede establecer la `TranslationX` y `TranslationY` propiedades en cualquier `VisualElement`, y que incluye `StackLayout`, Cuadrícula, e incluso Página y sus derivados. La transformación se aplica al elemento y todos los hijos de ese elemento.

Lo que podría no ser tan evidentes sin un poco de investigación es que `TranslationX` y `TranslationY` afecta sólo cómo es el elemento *prestado*. Estas propiedades hacen *no* afecta cómo el elemento es percibido dentro del sistema de diseño.

Por ejemplo, `VisualElement` define obtener propiedades de sólo el nombre X y Y indican que cuando un elemento se encuentra relacionada con su parente. los X y Y propiedades se establecen cuando un elemento se posiciona por su matriz, y en este ejemplo, la X y Y propiedades de Marco indicar la ubicación de la esquina superior izquierda de la Marco relativa a la esquina superior izquierda de la `StackLayout`. los X y Y propiedades hacen *no* cambiar cuando `TranslationX` y `TranslationY` se establecen. Además, el solo llegar Límites propiedad que combina X y Y junto con Anchura y Altura en una sola Rectángulo -*Eso no cambiar tampoco*. El sistema de diseño no se involucra cuando `TranslationX` y `TranslationY` son modificados.

¿Qué ocurre si se utiliza `TranslationX` y `TranslationY` para mover una Botón desde su posición original? ¿el Botón responder a los toques en su posición de diseño original o la nueva posición rendido? Se alegrará de saber que es esto último. `TranslationX` y `TranslationY` afectar tanto a la forma del elemento se hace y cómo responde a los toques. Usted verá esto en breve en un programa de ejemplo denominado `ButtonJump`.

Si necesita hacer algún movimiento extensivo de elementos alrededor de la página, puede que se pregunte si desea utilizar `AbsoluteLayout` y especificar las coordenadas de forma explícita o uso `TranslationX` y `TranslationY` para especificar las compensaciones. En términos de rendimiento, no hay realmente mucha diferencia. La ventaja de `TranslationX` y `TranslationY` es que se puede empezar con una posición establecida por `StackLayout` o Cuadrícula y luego mover los elementos relativos a esa posición.

Los efectos de texto

Una aplicación común de `TranslationX` y `TranslationY` es la aplicación de pequeños desplazamientos de los elementos que les cambian ligeramente de su posición de diseño. Esto a veces es útil si tiene varios elementos superpuestos en una sola célula Cuadrícula y la necesidad de cambiar de una manera que se asoma por debajo de otra.

Incluso puede utilizar esta técnica para efectos de texto comunes. El XAML de sólo `TextOffsets` programa pone tres pares de Etiqueta elementos en tres de una sola célula Cuadrícula diseños. El par de Etiqueta elementos en cada Cuadrícula son del mismo tamaño y mostrar el mismo texto:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " TextOffsets.TextOffsetsPage " >

< ContentPage.Padding >
    < OnPlatform x: TypeArguments = " Espesor "
        iOS = " 0, 20, 0, 0 " />
</ ContentPage.Padding >

< ContentPage.Resources >
    < ResourceDictionary >
        < Color x: Key = " color de fondo " > Blanco </ Color >
        < Color x: Key = " color de primer plano " > Negro </ Color >

        < Estilo Tipo de objetivo = " Cuadrícula " >
            < Setter Propiedad = " VerticalOptions " Valor = " CenterAndExpand " />
        </ Estilo >

        < Estilo Tipo de objetivo = " Etiqueta " >
            < Setter Propiedad = " Tamaño de fuente " Valor = " 72 " />
            < Setter Propiedad = " FontAttributes " Valor = " Negrita " />
            < Setter Propiedad = " HorizontalOptions " Valor = " Centrar " />
        </ Estilo >
    </ ResourceDictionary >
</ ContentPage.Resources >

< StackLayout Color de fondo = " { } StaticResource backgroundColor " >
    < Cuadrícula >
        < Etiqueta Texto = " Sombra "
            Color de texto = " { } StaticResource foregroundColor "
            Opacidad = " 0.5 "
            TranslationX = " 5 "
            TranslationY = " 5 " />

        < Etiqueta Texto = " Sombra "
            Color de texto = " { } StaticResource foregroundColor " />
```

```
</ Cuadricula >

< Cuadricula >
    < Etiqueta Texto = " Realizar "
        Color de texto = " {} StaticResource foregroundColor "
        TranslationX = " 2 "
        TranslationY = " 2 " />

    < Etiqueta Texto = " Realizar "
        Color de texto = " {} StaticResource backgroundColor " />
</ Cuadricula >

< Cuadricula >
    < Etiqueta Texto = " Grabar "
        Color de texto = " {} StaticResource foregroundColor "
        TranslationX = " -2 "
        TranslationY = " -2 " />

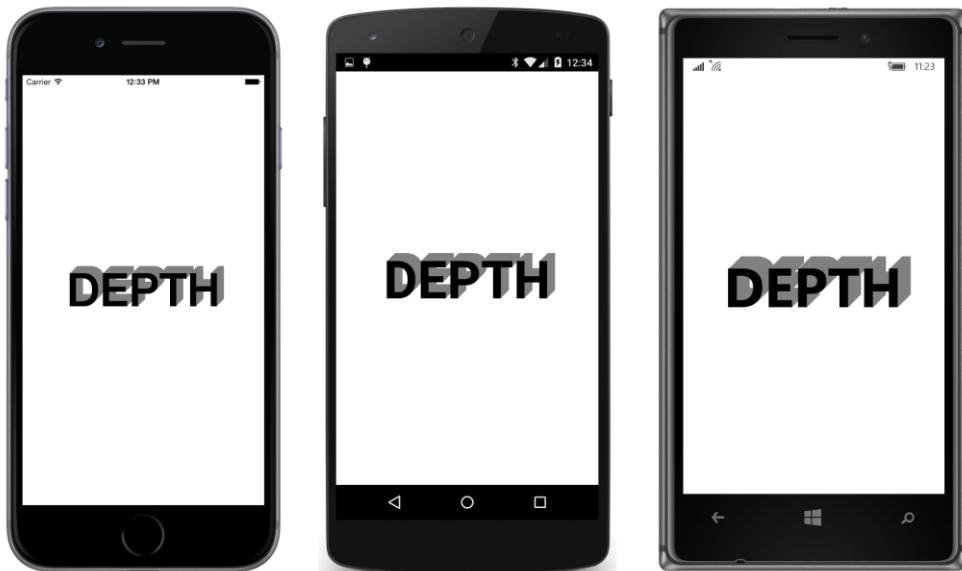
    < Etiqueta Texto = " Grabar "
        Color de texto = " {} StaticResource backgroundColor " />
</ Cuadricula >
</ StackLayout >
</ Pagina de contenido >
```

Normalmente, la primera Etiqueta en el Niños colección de la Cuadricula sería oscurecida por el segundo Etiqueta, pero TranslationX y TranslationY valores aplican sobre la primera Etiqueta permitir que sea parcialmente visible. Los mismos resultados técnica básica en tres efectos de texto diferentes: una gota de sombra, texto que aparece a ser levantado de la superficie de la pantalla, y el texto que parece que está cincelado en la pantalla:



Estos efectos dan un aspecto algo en 3D a las imágenes de otra manera plana. La ilusión óptica se basa en una convención que ilumina la pantalla desde la esquina superior izquierda. Por lo tanto, sombras se proyectan por debajo ya la derecha de los objetos planteados. La diferencia entre los efectos en relieve y grabado se debe enteramente a las posiciones relativas de la texto negro oculta y el texto en blanco en la parte superior. Si el texto negro es un poco por debajo ya la derecha, se convierte en la sombra de texto en blanco levantado. Si el texto negro está por encima ya la izquierda del texto en blanco, se convierte en una sombra de texto hundido por debajo de la superficie.

El siguiente ejemplo no es algo que querrá utilizar de forma regular, ya que requiere múltiples Etiquetas, pero la técnica se ilustra en la **BlockText** programa es útil si desea proporcionar un poco de "profundidad" a su texto:



los **BlockText** archivo XAML utiliza una sola célula Cuadrícula para mostrar el texto negro sobre un fondo blanco. El implícita (y extensa) Estilo definido para Etiqueta, sin embargo, especifica una Color de texto propiedad de Gris:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " BlockText.BlockTextPage " >

< Cuadrícula x: Nombre = " cuadrícula " >
    Color de fondo = " Blanco " >
< Grid.Resources >
    < ResourceDictionary >
        < Estilo Tipo de objetivo = " Etiqueta " >
            < Setter Propiedad = " Texto " Valor = " PROFUNDIDAD " />
            < Setter Propiedad = " Tamaño de fuente " Valor = " 72 " />
            < Setter Propiedad = " FontAttributes " Valor = " Negrita " />
            < Setter Propiedad = " Color de texto " Valor = " gris " />
            < Setter Propiedad = " HorizontalOptions " Valor = " Centrar " />
            < Setter Propiedad = " VerticalOptions " Valor = " Centrar " />
        </ Estilo >
    </ ResourceDictionary >
</ Grid.Resources >
</ Cuadrícula >
</ Página de contenido >
```

```

    </ ResourceDictionary >
</ Grid.Resources >

< Etiqueta Color de texto = "Negro" />

</ Cuadricula >
</ Página de contenido >

```

El constructor en el archivo de código subyacente agrega varios más Etiqueta elementos a la Cuadricula. los Estilo asegura que todos tengan las mismas propiedades (incluyendo el ser de color gris), pero cada uno de ellos está desplazado del Etiqueta en el archivo XAML:

```

pública clase parcial BlockTextPage : Página de contenido
{
    pública BlockTextPage ()
    {
        InitializeComponent ();

        para ( En t i = 0; i < Dispositivo .OnPlatform (12, 12, 18); i ++ )
        {
            grid.Children.Insert (0, nuevo Etiqueta
            {
                TranslationX = i,
                TranslationY = -i
            });
        }
    }
}

```

Aquí hay otro caso en el que Etiqueta elementos se solapan entre sí en el de una sola célula Cuadricula, pero ahora hay muchos más de ellos. El negro Etiqueta en el archivo XAML debe ser el *último* niño en el Chil-Dren colección para que sea en la parte superior de todas las demás. El elemento con el máximo TranslationX y TranslationY desplazamiento debe ser el *primer* niño en el Niños colección, por lo que debe estar en la parte inferior de la pila. Es por eso que cada sucesiva Etiqueta se inserta en el comienzo de la Niños colección.

Saltos y animaciones

los ButtonJump programa está destinado sobre todo para demostrar que no importa donde se mueve una Peritoneada en la pantalla mediante el uso de la traducción, la Botón todavía responde a los toques de la manera normal. Los centros de archivos XAML las Botón en el medio de la página (menos el relleno IOS en la parte superior):

```

< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " ButtonJump.ButtonJumpPage " >

    < ContentPage.Padding >
        < OnPlatform x: TypeArguments = " Espesor "
            iOS = " 0, 20, 0, 0 " />
    </ ContentPage.Padding >

    < ContentView >

```

```

< Botón Texto = "Me Toqué" 
    Tamaño de fuente = "Grande" 
    HorizontalOptions = "Centrar" 
    VerticalOptions = "Centrar" 
    hecho clic = "OnButtonClicked" />

</ ContentView >
</ Pagina de contenido >

```

Para cada llamada a la `OnButtonClicked` manejador, el archivo de código subyacente establece el `TranslationX` y `TranslationY` propiedades a los nuevos valores. Los nuevos valores se calculan al azar pero restringidos de modo que la Botón siempre permanece dentro de los bordes de la pantalla:

```

pública clase parcial ButtonJumpPage : Pagina de contenido
{
    Aleatorio = aleatorios nuevo Aleatorio ();

    pública ButtonJumpPage ()
    {
        InitializeComponent ();
    }

    vacío OnButtonClicked ( objeto remitente, EventArgs args )
    {
        Botón botón = ( Botón )remitente;
        Ver contenedor = ( Ver )Button.Parent;

        button.TranslationX = (random.NextDouble () - 0.5) * (container.Width - button.Width);
        button.TranslationY = (random.NextDouble () - 0.5) * (container.Height - button.Height);
    }
}

```

Por ejemplo, si el Botón es de 80 unidades de ancho y la ContentView es 320 unidades de ancho, la diferencia es de 240 unidades, que es de 120 unidades en cada lado de la Botón cuando está en el centro de la ContentView.

Los `NextDouble` método de `Aleatorio` devuelve un número entre 0 y 1, y restando 0,5 produce un número entre -0,5 y 0,5, lo que significa que `TranslationX` se establece en un valor aleatorio entre

- 120 y 120. Esos valores potencialmente posicionar el Botón hasta el borde de la pantalla, pero no más allá.

Manten eso en mente `TranslationX` y `TranslationY` son propiedades en lugar de métodos. Ellos no son acumulativos. Si se establece `TranslationX` a 100 y luego a 200, el elemento visual no se compensa con un total de 300 unidades a partir de su posición de diseño. El segundo `TranslationX` valor de 200 reemplaza en lugar de suma al valor inicial de 100.

A los pocos segundos que juegan con la `ButtonJump` Probablemente programa plantea una pregunta: ¿Puede esto ser animado? Puede el Botón deslizarse hasta el nuevo punto en lugar de simplemente ir allí?

Por supuesto. Hay varias maneras de hacerlo, incluyendo los métodos de animación Xamarin.Forms discutidos en el capítulo siguiente. El archivo XAML en el `ButtonGlide` programa es el mismo que el de `ButtonJump`, excepto que el Botón ahora tiene un nombre para que el programa puede crear fácilmente una referencia fuera de la hecho clic entrenador de animales:

```

< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " ButtonGlide.ButtonGlidePage " >

    < ContentPage.Padding >
        < OnPlatform x: TypeArguments = " Espesor "
            iOS = " 0, 20, 0, 0 " />
    </ ContentPage.Padding >

    < ContentView >
        < Botón x: Nombre = " botón "
            Texto = " Me Toquel "
            Tamaño de fuente = " Grande "
            HorizontalOptions = " Centrar "
            VerticalOptions = " Centrar "
            hecho clic = " OnButtonClicked " />
    </ ContentView >
</ Pagina de contenido >

```

El archivo de código subyacente procesa el clic de botón por el ahorro de varias piezas esenciales de información como campos: una Punto que indica la ubicación de partida obtenidos a partir de los valores actuales de TranslationX y TranslationY; un vector (que es también una Punto valor) calcula restando este punto de partida a partir de un punto de destino al azar; y la corriente Fecha y hora cuando el Botón se hace clic en:

```

pública clase parcial ButtonGlidePage : Pagina de contenido
{
    estático solo lectura Espacio de tiempo duración = Espacio de tiempo .FromSeconds (1);
    Aleatorio = aleatorios nuevo Aleatorio ();
    Punto punto de partida;
    Punto animationVector;
    Fecha y hora hora de inicio;

    pública ButtonGlidePage ()
    {
        InitializeComponent ();

        Dispositivo .StartTimer ( Espacio de tiempo .FromMilliseconds (16), OnTimerTick);
    }

    vacío OnButtonClicked ( objeto remitente, EventArgs args)
    {
        Botón botón = ( Botón )remitente;
        Ver contenedor = ( Ver ) Button.Parent;

        // El inicio de la animación son las propiedades de traducciones actuales.
        startPoint = nuevo Punto (Button.TranslationX, button.TranslationY);

        // El final de la animación es un punto aleatorio.
        doble EndX = (random.NextDouble () - 0.5) * (container.Width - button.Width);
        doble EndY = (random.NextDouble () - 0.5) * (container.Height - button.Height);

        // Se crea un vector desde el punto inicial hasta el punto final.
        animationVector = nuevo Punto (EndX - startPoint.X, EndY - startPoint.Y);

        // guardar la animación hora de inicio.
    }
}

```

```

        horaInicio = Fecha y hora .Ahora;
    }

    bool OnTimerTick ()
    {
        // obtener el tiempo transcurrido desde el inicio de la animación.
        Espacio de tiempo elapsedTime = Fecha y hora .now - fecha de inicio;

        // Normalizar el tiempo transcurrido de 0 a 1.
        doble t = Mates .MAX (0, Mates .min (1, elapsedTime.TotalMilliseconds /
            duration.TotalMilliseconds));

        // Calcular la nueva traducción basado en el vector de la animación.
        button.TranslationX = startPoint.X + t * animationVector.X;
        button.TranslationY = startPoint.Y + t * animationVector.Y;
        return true ;
    }
}

```

La devolución de llamada temporizador se llama cada 16 milisegundos. Eso no es un número arbitrario! pantallas de vídeo suelen tener una frecuencia de actualización de hardware de 60 veces por segundo. Por lo tanto, cada trama está activa durante aproximadamente 16 milisegundos. La estimulación de la animación a este ritmo es óptima. Una vez cada 16 milisegundos, la devolución de llamada calcula un tiempo transcurrido desde el **inicio de la animación y lo divide por la duración**. Eso es por lo general un valor llamado *t* (para *hora*) que oscila de 0 a 1 en el transcurso de la animación. Este valor se multiplica por el vector, y se añade el resultado a punto de partida. Ese es el nuevo valor de traducción *TranslationX* y *TranslationY*.

A pesar de que la devolución de llamada del temporizador se llama continuamente mientras se ejecuta la aplicación, la traducción *X* y *TranslationY* propiedades permanecen constantes cuando la animación se ha completado. Sin embargo, usted no tiene que esperar hasta que el Botón ha dejado de moverse antes de que pueda toque de nuevo. (Es necesario ser rápido, o puede cambiar la propiedad de duración a algo más.) La nueva animación se inicia desde la posición actual de la Botón y enteramente sustituye a la animación anterior.

Una de las ventajas del cálculo de un valor normalizado de *t* es que se vuelve bastante fácil de modificar ese valor para que la animación no tiene una velocidad constante. Por ejemplo, trate de añadir esta declaración después de que el cálculo inicial de *t*:

```
t = Mates .Sin (t * Mates .PI / 2);
```

Cuando el valor original de *t* es 0 en el inicio de la animación, el argumento de *Math.sin* es 0 y el resultado es 0. Cuando el valor original de *t* es 1, el argumento de *Math.sin* se pi / 2, y el resultado es 1. Sin embargo, los valores entre esos dos puntos no son lineales. Cuando el valor inicial de *t* es de 0,5, esta declaración recalcula *t* como el seno de 45 grados, que es 0,707. De manera que cuando la animación es un medio más, el Botón Ya ha subido un 70 por ciento de la distancia a su destino. En general, verá una animación que es más rápido al principio y más lenta hacia el final.

Vas a ver un par de enfoques diferentes a la animación en este capítulo. Aun cuando se haya familiarizado con el sistema de animación que proporciona Xamarin.Forms, a veces es útil para hacerlo usted mismo.

La transformada de escala

los VisualElement clase define una propiedad denominada Escala que se puede utilizar para cambiar el tamaño de un elemento prestado. Los Escala propiedad hace *no* afecten la disposición (como se demostrará en el **ButtonScaler** programa). Lo hace *no* afectará a la de sólo conseguir Anchura y Altura propiedades del elemento, o la presentación de sólo Límites propiedad que incorpora los Anchura y Altura valores. Los cambios en el

Escala propiedad, no *no* porque un SizeChanged evento que se dispara.

Escala afecta a las coordenadas de un elemento visual rendido, pero de una manera muy diferente de TranslationX y TranslationY. Las dos propiedades traducción agregan los valores de coordenadas, mientras que el Escala propiedad es multiplicativo. El valor por defecto de Escala es 1. Valores mayores que 1 aumentan el tamaño del elemento. Por ejemplo, un valor de 3 hace que el elemento de tres veces su tamaño normal. Valores menores que 1 disminuir el tamaño. UN Escala El valor 0 es legal, pero hace que el elemento sea invisible. Si está trabajando con Escala y el elemento parece haber desaparecido, comprobar si es de alguna manera conseguir una Escala valor de 0.

Los valores inferiores a 0 también son legales y que el elemento se puede girar 180 grados además de ser alterado en tamaño.

Usted puede experimentar con Escala ajustes mediante el **SimpleScaleDemo** programa. (El programa tiene una **Sencillo** prefijo, ya que no incluye el efecto de la anchorX y anchorY propiedades, que se discutirá en breve.) El XAML es similar a la **TranslationDemo** programa:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " SimpleScaleDemo.SimpleScaleDemoPage " >

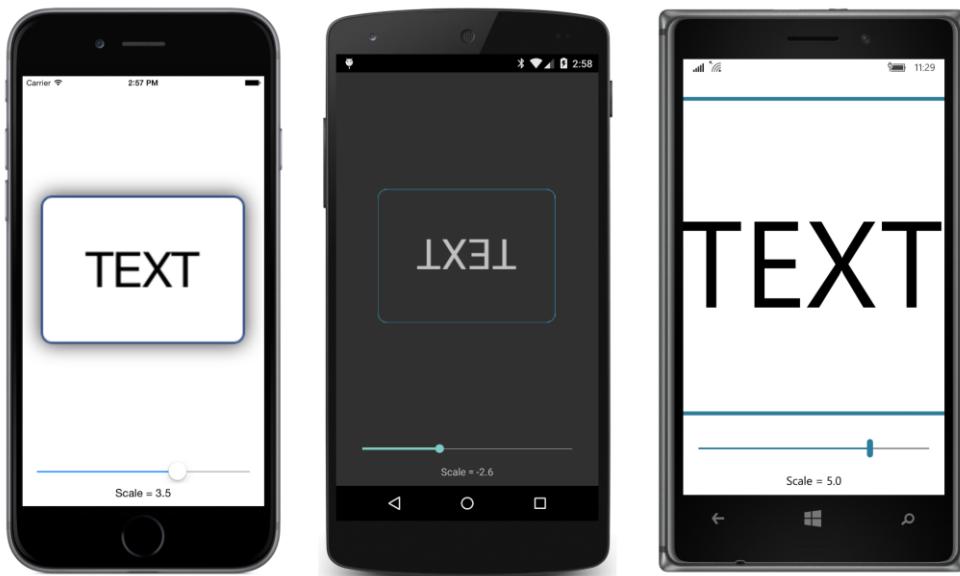
< StackLayout Relleno = " 20, 10 " >
    < Marco x: Nombre = " marco "
        HorizontalOptions = " Centrar "
        VerticalOptions = " CenterAndExpand "
        OutlineColor = " Acento " >

        < Etiqueta Texto = " TEXTO "
            Tamaño de fuente = " Grande " />
    </ Marco >

    < deslizador x: Nombre = " scaleSlider "
        Mínimo = " -10 "
        Máximo = " 10 "
        Valor = " {Binding Fuente = {x: Marco de Referencia},
                    Path = Escala} " />

    < Etiqueta Texto = " {Binding Fuente = {x: scaleSlider Referencia},
                    Path = Valor,
                    StringFormat = 'Escala = {0: F1}' } "
        HorizontalTextAlignment = " Centrar " />
</ StackLayout >
</ Página de contenido >
```

Aquí está en acción. Note el negativo Escala establecer en el teléfono Android:



En la pantalla móvil de Windows 10, el Marco ha sido reducido tan grande que no se puede ver sus lados izquierdo y derecho.

En la programación de la vida real, es posible que desee utilizar Escala para proporcionar un poco de retroalimentación a un usuario cuando una Botón se hace clic. Los Botón puede ampliar brevemente en tamaño y volver a bajar a la normalidad. Sin embargo, Escala no es la única manera de cambiar el tamaño de una Botón. También puede cambiar el Botón tamaño mediante el aumento y la disminución de la Tamaño de fuente propiedad. Estas dos técnicas son muy diferentes, sin embargo: La Escala propiedad no afecta la disposición, pero el Tamaño de fuente propiedad hace.

Esta diferencia se ilustra en el **ButtonScaler** programa. El archivo XAML se compone de dos Botón elementos intercalados entre dos pares de BoxView elementos:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns:x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " ButtonScaler.ButtonScalerPage " >

< StackLayout >
    <!-- Botón "Animate Escala" entre dos BoxViews. -->
    < BoxView Color = " Acento "
        HeightRequest = " 4 "
        VerticalOptions = " EndAndExpand " />

    < Botón Texto = " animar Escala "
        Tamaño de fuente = " Grande "
        Ancho del borde = " 1 "
        HorizontalOptions = " Centrar "
        hecho clic = " OnAnimateScaleClicked " />

    < BoxView Color = " Acento "
        HeightRequest = " 4 "
        VerticalOptions = " EndAndExpand " />
```

```

        HeightRequest = " 4 "
        VerticalOptions = " StartAndExpand " />

    <! - Botón "Animate Tamaño de Letra" entre dos BoxViews. ->
    < BoxView Color = " Acento "
        HeightRequest = " 4 "
        VerticalOptions = " EndAndExpand " />

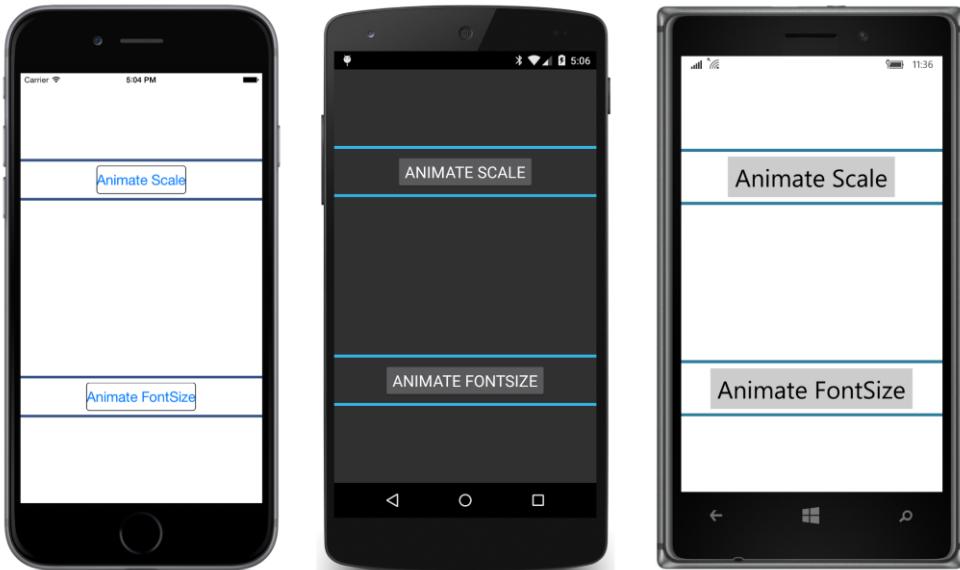
    < Botón Texto = " Animate Tamaño de Letra "
        Tamaño de fuente = " Grande "
        Ancho del borde = " 1 "
        HorizontalOptions = " Centrar "
        hecho clic = " OnAnimateFontSizeClicked " />

    < BoxView Color = " Acento "
        HeightRequest = " 4 "
        VerticalOptions = " StartAndExpand " />

</ StackLayout >
</ Página de contenido >

```

Esto es lo que la página se ve como normal:



El archivo de código subyacente implementa un método de animación algo generalizado. Está generalizado en el sentido de que los parámetros incluyen dos valores que indican el valor inicial y el valor final de la animación. Estos dos valores se llaman a menudo una *de* valor y una *a* valor. Los argumentos de animación también incluyen la duración de la animación y un método de devolución de llamada. El argumento para el método de devolución de llamada es un valor entre el “de” valor y la “a” valor y el método de llamada puede utilizar este valor para hacer lo que necesita para poner en práctica la animación.

Sin embargo, este método de animación no se generaliza en su totalidad. En realidad, calcula un valor de la

de valor a la *a* valor durante la primera mitad de la animación, y luego calcula un valor de la *a* valorar de nuevo a la *de* valor durante la segunda mitad de la animación. Esto a veces se llama una *revertir* animación.

El método se llama **AnimateAndBack**, y se utiliza una **Task.Delay** llamar a pasarse por la animación y una **.RED Stopwatch** objeto para determinar el tiempo transcurrido:

```

pública clase parcial ButtonScalerPage : Pagina de contenido
{
    pública ButtonScalerPage ()
    {
        InitializeComponent ();
    }

    vacío OnAnimateScaleClicked ( objeto remitente, EventArgs args )
    {
        Botón botón = ( Botón )remitente;
        AnimateAndBack ( 1, 5, Espacio de tiempo .FromSeconds ( 3 ), ( doble valor ) =>
        {
            button.Scale = valor;
        });
    }

    vacío OnAnimateFontSizeClicked ( objeto remitente, EventArgs args )
    {
        Botón botón = ( Botón )remitente;

        AnimateAndBack ( button.FontSize, 5 * button.FontSize,
                         Espacio de tiempo .FromSeconds ( 3 ), ( doble valor ) =>
        {
            button.FontSize = valor;
        });
    }

    vacío asíncrono AnimateAndBack ( doble fromValue, doble valor,
                                    Espacio de tiempo duración, Acción < doble > Devolución de llamada )
    {
        Cronógrafo cronómetro = nuevo Cronógrafo ();
        doble t = 0;
        stopWatch.Start ();

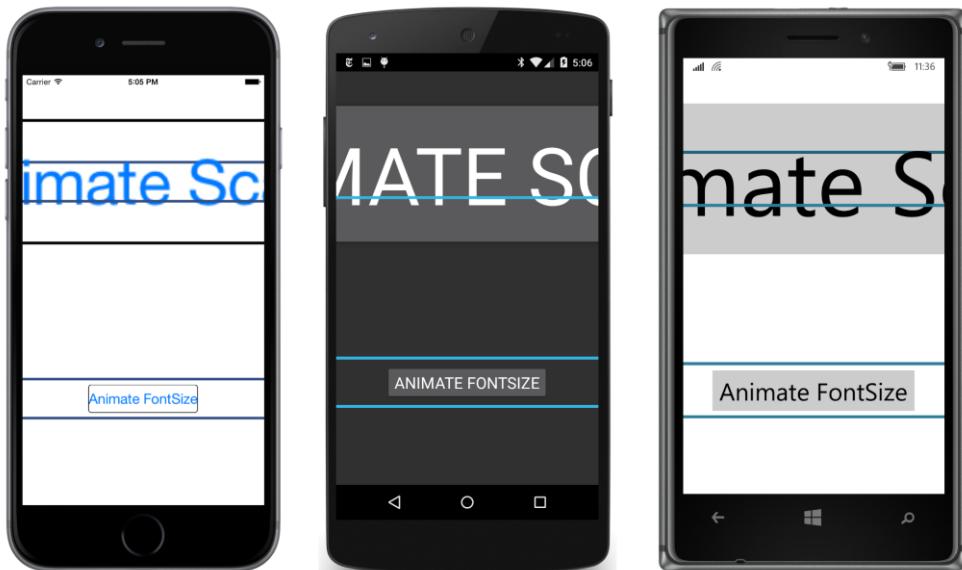
        mientras ( T <1 )
        {
            doble tReversing = 2 * ( t <0,5 t: 1 - t? );
            devolución de llamada ( + fromValue ( toValue - fromValue ) * tReversing );
            esperar Tarea .Delay ( 16 );
            t = stopWatch.ElapsedMilliseconds / duration.TotalMilliseconds;
        }

        stopWatch.Stop ();
        de devolución de llamada ( fromValue );
    }
}

```

los hecho clic manejadores de los dos botones de cada puesta en marcha de una animación independiente. los hecho clic controlador para el primer Botón anima su Escala propiedad de 1 a 5 y de vuelta, mientras que el hecho clic controlador para el segundo Botón anima su Tamaño de fuente propiedad con un factor de escala de 1 a 5 y de vuelta otra vez.

Aquí está la animación de la Escala A mitad de la propiedad sobre:



Como se puede ver, la escala de la Botón no toma en cuenta de cualquier cosa que pueda estar en la pantalla, y en las pantallas móviles iOS y Windows 10 en realidad se puede ver a través de las áreas transparentes de la Botón a la cima BoxView elementos, mientras que el Android opaca Botón totalmente oculta que la parte superior BoxView. los BoxView por debajo de la parte superior Botón en realidad se encuentra en la parte superior de la Botón y es visible en las tres plataformas.

Un aumento de la animación Tamaño de fuente la propiedad se maneja de manera diferente en las tres plataformas:



En iOS, la Botón El texto se trunca en el medio y la Botón sigue siendo la misma altura. En Android, la Botón envolturas de texto y la ampliada Botón empuja los dos BoxView elementos aparte. El tiempo de ejecución de Windows Botón También trunca el texto pero de una manera diferente a la de iOS, y al igual que Android, el aumento Botón La altura también empuja los dos BoxView elementos de distancia.

la animación de la Escala propiedad no afecta el diseño, pero la animación de la Tamaño de fuente propiedad obviamente afecta a la disposición.

El pequeño sistema de animación implementado en **ButtonScaler** puede animar los dos botones de forma independiente y simultánea, pero, sin embargo, tiene un defecto grave. Intente tocar una Botón mientras que Botón Actualmente se está animada. Una nueva animación se pondrá en marcha para ese Botón, y los dos animaciones interferirán entre sí.

Hay un par de maneras de solucionar este problema. Una posibilidad es incluir una CancellationToken valor como un argumento a la `AnimateAndBack` método para que el método puede ser cancelada. (Puede pasar esta misma CancellationToken valor a la Task.Delay llamar.) Esto permitiría a la hecho clic controlador para el Botón para cancelar cualquier animación en curso antes de que comience una nueva.

Otra opción es que `AnimateAndBack` para devolver una Tarea objeto. Esto permite que el hecho clic manejador de los botones para utilizar el esperar operador `AnimateAndBack`. los Botón puede desactivar fácilmente en sí antes de llamar `AnimateAndBack` y volver a habilitar a sí mismo cuando `AnimateAndBack` ha completado la animación.

En cualquier caso, si se desea implementar la retroalimentación al usuario con un breve aumento y disminución de Botón tamaño, es más seguro y más eficiente para animar Escala más bien que Tamaño de fuente. Verás otras técnicas para hacer esto en el siguiente capítulo de la animación, y en el capítulo 23, "disparadores y comportamientos."

Otro uso de la Escala la propiedad es el encolado un elemento para ajustarse al espacio disponible. Se puede recordar la **FitToSizeClock** programa hacia el final del capítulo 5, "Tratar con los tamaños." Usted puede hacer algo muy similar con el Escala propiedad, pero no será necesario para hacer estimaciones o cálculos recursivos.

El archivo XAML de la **ScaleToSize** programa contiene una Etiqueta falta algún texto y también la falta de un Escala establecer para que la Etiqueta mayor:

```
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns:x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " ScaleToSize.ScaleToSizePage "
    SizeChanged = " OnSizeChanged ">

    < Etiqueta x: Nombre = " etiqueta "
        HorizontalOptions = " Centrar "
        VerticalOptions = " Centrar "
        SizeChanged = " OnSizeChanged " />

</ Pagina de contenido >
```

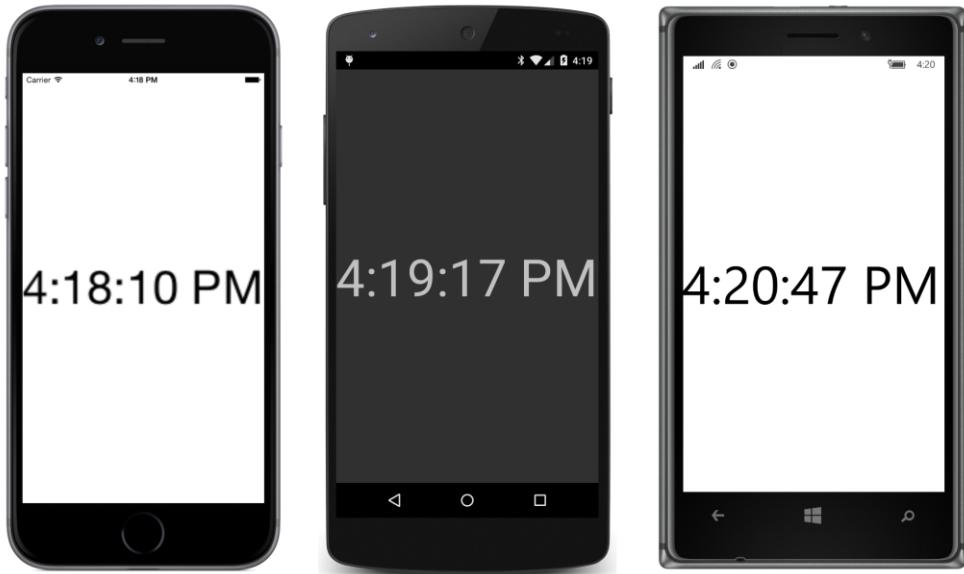
Ambos Pagina de contenido y el Etiqueta tener SizeChanged manipuladores instalados, y que ambos utilizan el mismo controlador. Este controlador se limita a establecer la Escala propiedad de la Etiqueta al mínimo de la anchura y la altura de la página dividida por la anchura y la altura de la Etiqueta:

```
p\u00f3blico clase parcial ScaleToSizePage : Pagina de contenido
{
    p\u00f3blico ScaleToSizePage ()
    {
        InitializeComponent ();
        UpdateLoop ();
    }

    vacio asincrono UpdateLoop ()
    {
        mientras ( cierto )
        {
            label.text = Fecha y hora .Now.ToString ( "T" );
            esperar Tarea .Delay (1000);
        }
    }

    vacio OnSizeChanged ( objeto remitente, EventArgs args)
    {
        label.Scale = Mates .min (Ancho / label.Width, Altura / label.Height);
    }
}
```

Debido a que el establecimiento de la Escala la propiedad no se dispara otra SizeChanged caso, no hay peligro de provocar un bucle recursivo sin fin. Pero un bucle infinito real utilizando Task.Delay mantiene el Etiqueta actualizado con el tiempo actual:



Por supuesto, convirtiendo a los lados del teléfono hace que la Etiqueta mayor:



Y aquí se puede detectar una pequeña diferencia en la aplicación de la Escala propiedad en iOS en comparación con Android y el tiempo de ejecución de Windows. En Android y Windows, el texto resultante se ve como si hubiera sido dibujado con una fuente grande. Sin embargo, el texto en la pantalla de iOS se ve un poco borrosa. Esta falta de claridad se produce cuando el sistema operativo rasteriza la preescalado Etiqueta, lo que significa que el sistema operativo lo convierte en un mapa de bits. El mapa de bits se ampliada, basada en la Escala ajuste.

El anclaje de la escala

Como usted ha experimentado con el Escala propiedad, usted probablemente ha notado que cualquier expansión del elemento visual se produce hacia fuera desde el centro del elemento, y si achica un elemento visual a nada, se contrae hacia el centro también.

Aquí hay otra manera de pensar en ello: El punto en el centro del elemento visual permanece en el mismo lugar independientemente de la configuración de la Escala propiedad.

Si está utilizando el Escala propiedad para expandir una Botón para la retroalimentación visual o para adaptarse a un elemento visual dentro de un espacio determinado, que probablemente es precisamente lo que quiere. Sin embargo, para algunas otras aplicaciones, es posible que en vez preferiría que otro punto permanece en el mismo lugar con los cambios en el

Escala propiedad. Tal vez desea que la esquina superior izquierda del elemento visual a permanecer en el mismo lugar y para la expansión del objeto que se produzca hacia la derecha y abajo.

Se puede controlar el centro de la escala con el anchorX y anchorY propiedades. Estas propiedades son de tipo doble y son en relación con el elemento que se está transformado. Un anchorX valor de 0 indica el lado izquierdo del elemento, y un valor de 1 es el lado derecho del elemento. Un anchorY valor de 0 es la parte superior y 1 es la parte inferior. Los valores predeterminados son 0,5, que es el centro. Configuración de las propiedades a 0 permite la escalabilidad que es relativa a la esquina superior izquierda del elemento.

También puede establecer las propiedades a valores inferiores a 0 o mayor que 1, en cuyo caso el centro de la escala está fuera de los límites del elemento.

Como verá, la anchorX y anchorY propiedades también afectan a la rotación. La rotación se produce alrededor de un punto particular, llamado el *centro de rotación*, y estas dos propiedades establecen ese punto con respecto al elemento que se hace girar.

los AnchoredScaleDemo programa te permite experimentar con anchorX y anchorY ya que afectan a la Escala propiedad. Los archivos XAML contiene dos paso a paso vistas que le permiten cambiar la anchorX y anchorY propiedades de -1 a 2 en incrementos de 0,25:

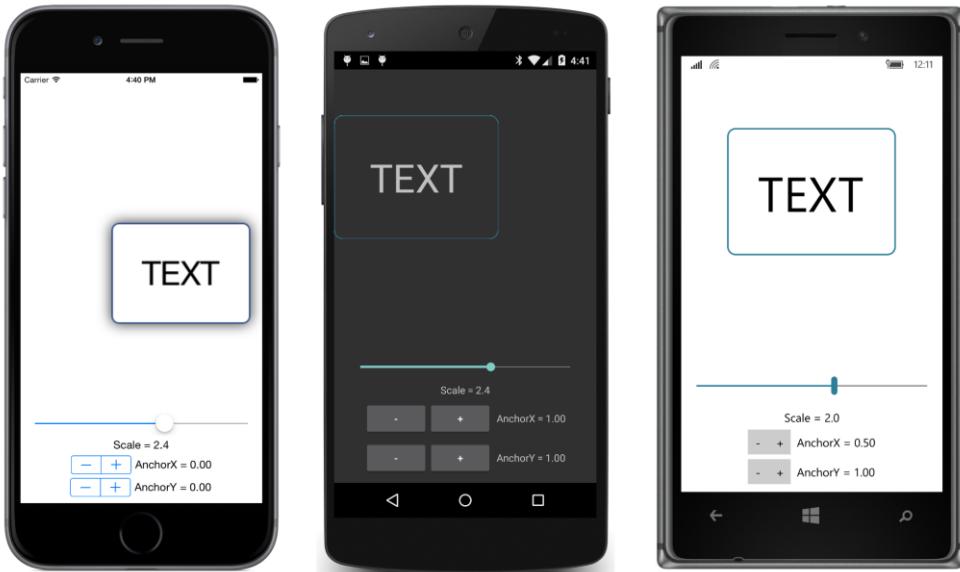
```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns:x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " AnchoredScaleDemo.AnchoredScaleDemoPage " >

    < StackLayout Relleno = " 20, 10 " >
        < Marco x: Nombre = " marco "
            HorizontalOptions = " Centrar "
            VerticalOptions = " CenterAndExpand "
            OutlineColor = " Acento " >
            < Etiqueta Texto = " TEXTO "
                Tamaño de fuente = " Grande " />
        </ Marco >

        < deslizador x: Nombre = " scaleSlider "
            Mínimo = " -10 "
            Máximo = " 10 "
            Valor = " {Binding Fuente = {x: Marco de Referencia},
                Path = Escala} " />
    </ StackLayout >
</ Página de contenido >
```

```
< Etiqueta Texto = " {Binding Fuente = {x: scaleSlider Referencia}},  
    Path = Valor,  
    StringFormat = 'Escala = {0: F1}' "  
    HorizontalTextAlignment = " Centrar " />  
  
< StackLayout Orientación = " Horizontal "  
    HorizontalOptions = " Centrar " >  
    < paso a paso x: Nombre = " anchorXStepper "  
        Mínimo = " -1 "  
        Máximo = " 2 "  
        Incremento = " 0.25 "  
        Valor = " {Binding Fuente = {x: Marco de Referencia}},  
            Path = anchorX } " />  
  
    < Etiqueta Texto = " {Binding Fuente = {x: anchorXStepper Referencia}},  
        Path = Valor,  
        StringFormat = 'anchorX = {0: F2}' "  
        VerticalOptions = " Centrar " />  
    </ StackLayout >  
  
< StackLayout Orientación = " Horizontal "  
    HorizontalOptions = " Centrar " >  
    < paso a paso x: Nombre = " anchorYStepper "  
        Mínimo = " -1 "  
        Máximo = " 2 "  
        Incremento = " 0.25 "  
        Valor = " {Binding Fuente = {x: Marco de Referencia}},  
            Path = anchorY } " />  
  
    < Etiqueta Texto = " {Binding Fuente = {x: anchorYStepper Referencia}},  
        Path = Valor,  
        StringFormat = 'anchorY = {0: F2}' "  
        VerticalOptions = " Centrar " />  
    </ StackLayout >  
    </ StackLayout >  
</ Página de contenido >
```

Aquí hay algunas capturas de pantalla que muestran (de izquierda a derecha) descamada lo que es relativa a la esquina superior izquierda, con relación a la esquina inferior derecha, y con relación a la parte inferior central:



Si está familiarizado con la programación iOS, que sabe sobre el similares Anchorpoint propiedad. En iOS, esta propiedad afecta tanto el posicionamiento y el centro de transformación. En Xamarin.Forms, la anchorX y anchorY propiedades especifican sólo el centro de transformación.

Este significa que la aplicación de IOS de Xamarin.Forms debe compensar la diferencia entre Anchorpoint y el anchorX y anchorY propiedades, y en la última versión de Xamarin.Forms disponibles ya que esta edición se va a imprimir, que la compensación no está funcionando del todo bien.

Para ver el problema, implemente el AnchoredScaleDemo programa para un simulador de iPhone o iPhone. Salir Escala ajustado a su valor por defecto de 1, pero establecer tanto anchorX y anchorY a 1. El Marco con el Etiqueta no debe moverse desde el centro de su ranura en el StackLayout porque el anchorX y Un-Chory propiedades sólo deberían afectar el centro de la escala y la rotación.

Ahora cambia la orientación del teléfono o un simulador de vertical a horizontal. los Marco ya no está centrado. Ahora cambiarlo de nuevo a vertical. No vuelve a su posición centrada originales.

Este problema afecta a todos los programas en este capítulo (y el siguiente capítulo) que utilizan los valores no predeterminados de anchorX y AnchorY. A veces los programas de ejemplo de estos capítulos establecen anchorX y anchorY después de que un elemento ha sido redimensionada para tratar de evitar el problema, pero siempre y cuando el teléfono se puede cambiar la orientación de vertical a horizontal, el problema no puede ser eludido, y no hay nada que una aplicación puede hacer para compensar el problema.

La rotación de la transformada

los Rotación propiedad hace girar un elemento visual en la superficie de la pantalla. Selecciona el Rotación propiedad a un ángulo en grados (no radianes). Los ángulos positivos rotan el elemento de las agujas del reloj. Se puede establecer Rotación a los ángulos de menos de 0 o mayor que 360. El ángulo de rotación real es el valor de la Rotación propiedad de módulo 360. El elemento se hace girar alrededor de un punto con respecto a sí mismo especificado con el anchorX y anchorY propiedades.

los **PlaneRotationDemo** programa le permite experimentar con estas tres propiedades. El archivo XAML es muy similar a la **AnchoredScaleDemo** programa:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " PlaneRotationDemo.PlaneRotationDemoPage " >

< StackLayout Relleno = " 20, 10 " >
    < Marco x: Nombre = " marco " 
        HorizontalOptions = " Centrar "
        VerticalOptions = " CenterAndExpand "
        OutlineColor = " Acento " >
        < Etiqueta Texto = " TEXTO "
            Tamaño de fuente = " Grande " />
        </ Marco >

        < deslizador x: Nombre = " rotationSlider "
            Máximo = " 360 "
            Valor = " {Binding Fuente = {x: Marco de Referencia},
                Path = Rotación} " />

        < Etiqueta Texto = " {Binding Fuente = {x: rotationSlider Referencia},
                Path = Valor,
                StringFormat = 'Rotación = {0: F0}' } "
            HorizontalTextAlignment = " Centrar " />

    < StackLayout Orientación = " Horizontal "
        HorizontalOptions = " Centrar " >
        < paso a paso x: Nombre = " anchorXStepper "
            Mínimo = " -1 "
            Máximo = " 2 "
            Incremento = " 0.25 "
            Valor = " {Binding Fuente = {x: Marco de Referencia},
                Path = anchorX} " />

        < Etiqueta Texto = " {Binding Fuente = {x: anchorXStepper Referencia},
                Path = Valor,
                StringFormat = 'anchorX = {0: F2}' } "
            VerticalOptions = " Centrar " />
    </ StackLayout >

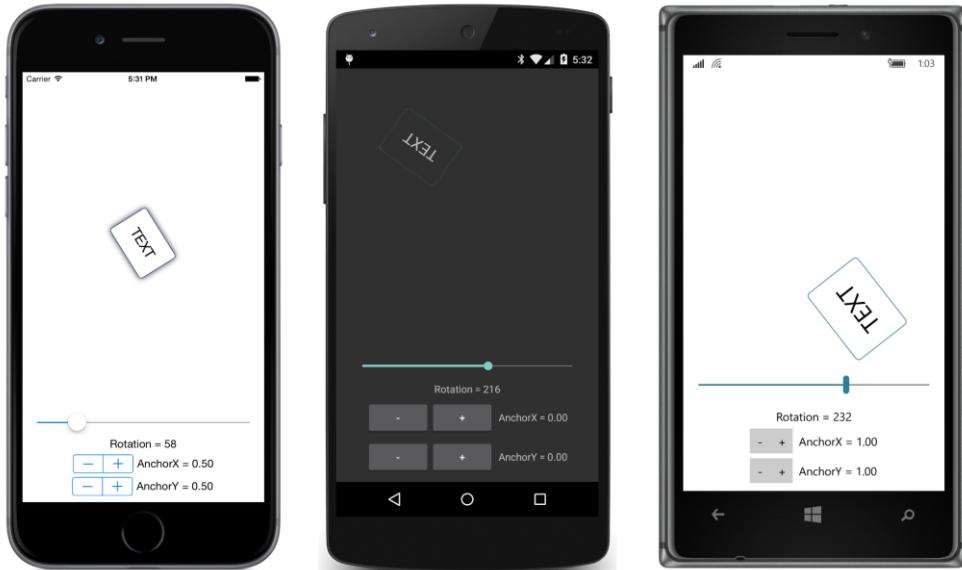
    < StackLayout Orientación = " Horizontal "
        HorizontalOptions = " Centrar " >
        < paso a paso x: Nombre = " anchorYStepper "
            Mínimo = " -1 "
            Máximo = " 2 "
            Incremento = " 0.25 "
            Valor = " {Binding Fuente = {x: Marco de Referencia},
                Path = anchorY} " />
```

```

Mínimo = "-1"
Máximo = "2"
Incremento = "0.25"
Valor = "(Binding Fuente = {x: Marco de Referencia},
Path = anchorY)"/>
< Etiqueta Texto = "(Binding Fuente = {x: anchorYStepper Referencia}),
Path = Valor,
StringFormat = 'anchorY = {0: F2}')"
VerticalOptions = "Centrar" />
</ StackLayout >
</ StackLayout >
</ Página de contenido >

```

Aquí hay varias combinaciones de Rotación ángulos y centros de rotación:



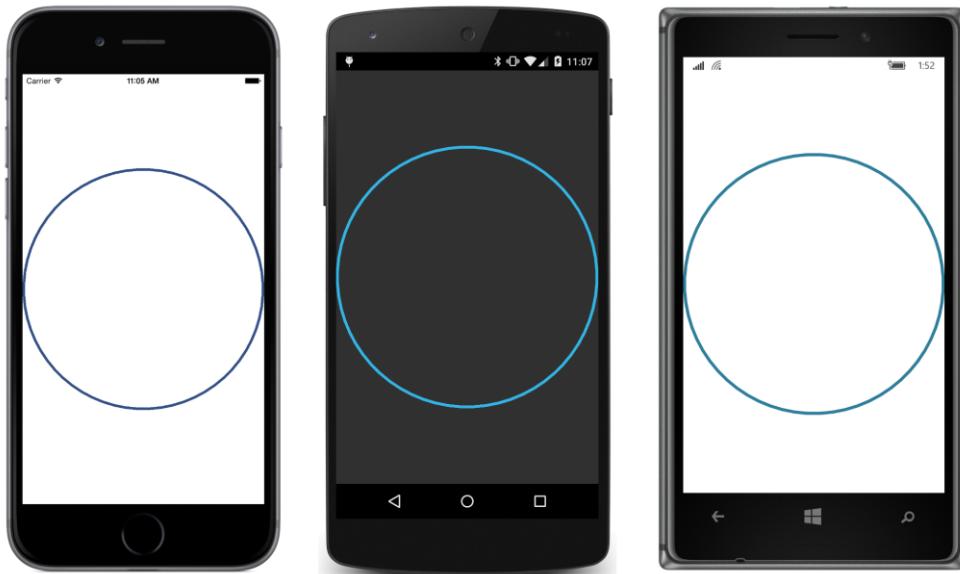
La pantalla de iOS muestra rotación alrededor del centro del elemento (que siempre es seguro en iOS a pesar de la anchorX y anchorY bug), mientras que la rotación de la pantalla de Android es la vuelta de la esquina superior izquierda, y la rotación de la pantalla en Windows Mobile 10 se centra en la esquina inferior derecha.

efectos de texto girado

La rotación es divertido. Es más divertido cuando se anima rotación (como se verá en el siguiente capítulo), pero es divertido, incluso con imágenes estáticas.

Varios de los ejemplos de rotación en este capítulo y el siguiente implican la organización de los elementos visuales en un círculo, así que vamos a empezar por intentar mostrar un simple círculo. Por supuesto, sin un sistema de gráficos reales en Xamarin.Forms, tendremos que ser creativos y construimos este círculo con BoxView. Si utiliza muchos pequeños BoxView elementos y organizar de manera adecuada, debería ser posible crear algo

que se parece a un círculo redonda y lisa, como este:



Cada círculo se compone de 64 BoxView elementos, cada uno de los cuales es de 4 unidades de espesor. Estos dos valores se definen como constantes en el código de sólo **BoxViewCircle** programa:

```

pública clase BoxViewClockPage : Pagina de contenido
{
    const int COUNT = 64;
    const doble ESPESOR = 4;

    pública BoxViewClockPage ()
    {
        AbsoluteLayout AbsoluteLayout = nuevo AbsoluteLayout ();
        Content = AbsoluteLayout;

        para ( En t index = 0; índice <COUNT; índice++)
        {
            absoluteLayout.Children.Add ( nuevo BoxView
            {
                color = Color .Acento,
            });
        }

        absoluteLayout.SizeChanged += (remitente, args) =>
        {
            Punto centro = nuevo Punto (AbsoluteLayout.Width / 2, absoluteLayout.Height / 2);
            doble radio = Mates .min (absoluteLayout.Width, absoluteLayout.Height) / 2;
            doble circunferencia = 2 * Mates .Pi * radio;
            doble longitud = circunferencia / COUNT;

            para ( En t index = 0; índice <absoluteLayout.Children.Count; índice++)

```

```

    {
        BoxView boxView = (BoxView) AbsoluteLayout.Children [indice];

        // Posición cada BoxView en la parte superior.
        AbsoluteLayout .SetLayoutBounds (boxView,
            nuevo Rectángulo (Center.X - longitud / 2,
                center.Y - radio,
                longitud,
                ESPESOR));

        // Establecer el anchorX y propiedades anchorY rotación es tan
        // alrededor del centro de la AbsoluteLayout.
        boxView.AnchorX = 0,5;
        boxView.AnchorY = radio / ESPESOR;

        // Establecer una rotación único para cada BoxView.
        boxView.Rotation = índice * 360,0 / COUNT;
    }
}
}

```

Todos los cálculos se producen en el `SizeChanged` manejador de la `AbsoluteLayout`. El mínimo de la anchura y la altura de la `AbsoluteLayout` es el radio de un círculo. Sabiendo que el radio permite el cálculo de una circunferencia, y por lo tanto una longitud para cada individuo `BoxView`.

los para posiciones de bucle cada BoxView en el mismo lugar: en la parte superior central del círculo. Cada BoxView debe entonces ser girada alrededor del centro del círculo. Esto requiere el establecimiento de una anchorY propiedad que corresponde a la distancia desde la parte superior de la BoxView al centro del círculo. Esa distancia es la radio valor, pero debe ser en unidades de la BoxView altura, lo que significa que radio debe ser dividido por ESPESOR.

Hay una forma alternativa para posicionar y rotar cada BoxView que no requiere ajuste de la `anchorX` y `anchorY` propiedades. Este enfoque es mejor para iOS. Los para bucle comienza calculando X y y valores correspondientes al centro de cada BoxView alrededor del perímetro del círculo. Estos cálculos requieren el uso de las funciones seno y coseno con una radio valor que compensa el espesor de la BoxView:

```

para ( En t index = 0; indice <absoluteLayout.Children.Count; índice ++)

{
    BoxView boxView = ( BoxView ) AbsoluteLayout.Children [índice];

    // encontrar el punto en el centro de cada BoxView posicionado.

    doble radianes = índice * 2 * Mates .PI / COUNT;
    doble x = center.X + (radio - ESPESOR / 2) * Mates .Sin (radianes);
    doble y = center.Y - (radio - ESPESOR / 2) * Mates .Cos (radianes);

    // Posición cada BoxView en ese punto.

    AbsoluteLayout .SetLayoutBounds (boxView,
        nuevo Rectángulo (X - longitud / 2,
                           y - ESPESOR / 2,
                           longitud,
                           longitud));
}

```

```

        ESPESOR));

// Establecer una rotación único para cada BoxView.
boxView.Rotation = indice * 360,0 / COUNT;
}

```

los X y y valores indican la posición deseada para el centro de cada BoxView, mientras AbsoluteLayout.out.SetLayoutBounds requiere la ubicación de la esquina superior izquierda de cada BoxView, por lo que estos X y y los valores se ajustan para que la diferencia cuando se utiliza con SetLayoutBounds. Cada BoxView Entonces se hace girar alrededor de su propio centro.

Ahora vamos a girar un poco de texto. los **RotatedText** programa es implementado en su totalidad en XAML:

```

< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " RotatedText.RotatedTextPage " >

    < Cuadricula >
        < Grid.Resources >
            < ResourceDictionary >
                < Estilo Tipo de objetivo = " Etiqueta " >
                    < Setter Propiedad = " Texto " Valor = " GIRAR " />
                    < Setter Propiedad = " Tamaño de fuente " Valor = " 32 " />
                    < Setter Propiedad = " Grid.Column " Valor = " 1 " />
                    < Setter Propiedad = " VerticalOptions " Valor = " Centrar " />
                    < Setter Propiedad = " HorizontalOptions " Valor = " comienzo " />
                    < Setter Propiedad = " anchorX " Valor = " 0 " />
                </ Estilo >
            </ ResourceDictionary >
        </ Grid.Resources >

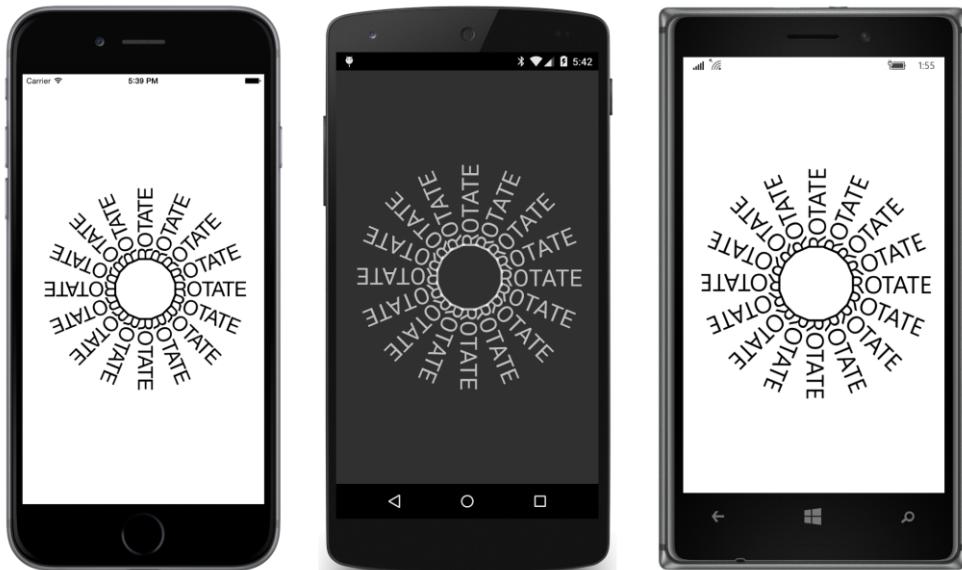
        < Etiqueta Rotación = " 0 " />
        < Etiqueta Rotación = " 22,5 " />
        < Etiqueta Rotación = " 45 " />
        < Etiqueta Rotación = " 67,5 " />
        < Etiqueta Rotación = " 90 " />
        < Etiqueta Rotación = " 112,5 " />
        < Etiqueta Rotación = " 135 " />
        < Etiqueta Rotación = " 157,5 " />
        < Etiqueta Rotación = " 180 " />
        < Etiqueta Rotación = " 202,5 " />
        < Etiqueta Rotación = " 225 " />
        < Etiqueta Rotación = " 246,5 " />
        < Etiqueta Rotación = " 270 " />
        < Etiqueta Rotación = " 292,5 " />
        < Etiqueta Rotación = " 315 " />
        < Etiqueta Rotación = " 337,5 " />
    </ Cuadricula >
</ Pagina de contenido >

```

El programa consiste en 16 Etiqueta elementos en una Cuadricula con una implícita Estilo el establecimiento de seis propiedades, incluyendo el Texto y Tamaño de fuente. Aunque esto Cuadricula podría parecer ser sólo una sola célula, que es en realidad una de dos columnas Cuadricula porque el Estilo establece el Grid.Column propiedad de cada Etiqueta a 1, que es la segunda columna. los Estilo centros de cada uno Etiqueta verticalmente dentro de la segunda columna y lo inicia en

la izquierda de esa columna, que es el centro de la página. Sin embargo, el texto comienza con varios espacios en blanco, por lo que parece empezar un poco a la derecha del centro de la página.

los Estilo concluye estableciendo la anchorX valor a 0, que establece el centro de rotación al centro de la vertical del borde izquierdo de cada Etiqueta. Cada Etiqueta a continuación, crear una única Rotación ajuste:



Obviamente, los espacios anteriores a la cadena "GIRAR" se eligen de modo que las barras verticales de la R se combinan para formar un polígono de 16 lados que parece casi como un círculo.

También puede girar letras individuales en una cadena de texto si cada letra es una separada Etiqueta elemento. Se empieza por la colocación de éstos Etiqueta elementos de una AbsoluteLayout y luego aplicar una Rotación propiedad para hacer que parezca como si las letras siguen un camino lineal en particular. los CircularText programa organiza estas letras en un círculo.

CircularText es un programa de código de sólo y es similar a la alternativa **BoxViewCircle** algoritmo. El constructor es responsable de la creación de todo el individuo Etiqueta elementos y añadirlos a la Chil-Dren colección de la AbsoluteLayout. Sin posicionamiento o la rotación se realiza durante el constructor porque el programa todavía no sabe lo grande que estos individuos Etiqueta elementos son, o cuán grande es el AbsoluteLayout es:

```
clase pública CircularTextPage : Pagina de contenido
{
    AbsoluteLayout AbsoluteLayout;
    Etiqueta [] etiquetas;

    público CircularTextPage ()
    {
        // crear el AbsoluteLayout.
        AbsoluteLayout = nuevo AbsoluteLayout ();
    }
}
```

```
absoluteLayout.SizeChanged += (remitente, args) =>
{
    LayOutLabels ();
};

Content = AbsoluteLayout;

// crear las etiquetas.
cuerda text = "Xamarin.Forms me dan ganas de codificar más con";
etiquetas = nuevo Etiqueta [text.length];
doble fontSize = 32;
En t countSized = 0;

para ( En t index = 0; índice <text.length; índice++)
{
    carbonizarse ch = texto [índice];

    Etiqueta etiqueta = nuevo Etiqueta
    {
        Text = CH == "?" ? ":" Ch.ToString (),
        Opacidad = CH == "?" ? 0: 1,
        FontSize = fontSize,
    };
    label.SizeChanged += (remitente, args) =>
    {
        Si (++ countSized> = labels.Length)
            LayOutLabels ();
    };

    etiquetas [índice] = etiqueta;
    absoluteLayout.Children.Add (etiqueta);
}
}

vacio LayOutLabels ()
{
    // Calcular la anchura total de las etiquetas.
    doble totalWidth = 0;

    para cada ( Etiqueta etiqueta en etiquetas)
    {
        totalWidth += label.Width;
    }

    // A partir de eso, conseguir un radio de la circunferencia al centro de las etiquetas.
    doble radio = totalWidth / 2 / Mates .PI + etiquetas [0] .height / 2;
    Punto centro = nuevo Punto (AbsoluteLayout.Width / 2, absoluteLayout.Height / 2);
    doble ángulo = 0;

    para ( En t index = 0; índice <labels.Length; índice++)
    {
        Etiqueta etiqueta = etiquetas [índice];

        // Establecer la posición de la etiqueta.
        doble x = center.X + radio * Mates .Sin (ángulo) - label.Width / 2;
```

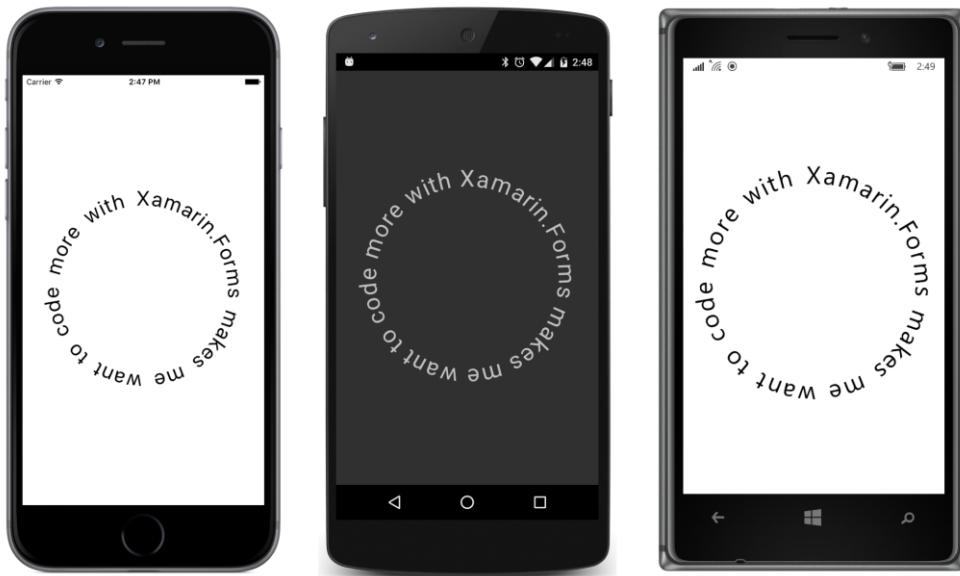
Observe el código que crea cada uno Etiqueta elemento: si el carácter de la cadena de texto original es un espacio, el Texto propiedad de la Etiqueta se le asigna un guión, pero el Opacidad propiedad se establece en 0 para que el tablero es invisible. Este es un pequeño truco para solucionar un problema que se presentó en las plataformas de Windows en tiempo de ejecución: Si el Etiqueta contiene sólo un espacio, a continuación, la anchura de la Etiqueta se calcula como cero y todas las palabras corren juntos.

Toda la acción ocurre en el LayOutLabels método. Este método se llama a partir de dos Tamaño-cambiado manipuladores expresan como funciones lambda en el constructor. los SizeChanged controlador para el AbsoluteLayout se llama poco después de iniciar el programa o cuando la orientación del teléfono cambia. los SizeChanged controlador para el Etiqueta Elementos mantiene un registro de cuántas se han dimensionado hasta ahora, y sólo las llamadas LayOutLabels cuando están todos listos.

los LayOutLabels método calcula la anchura total de todo el Etiqueta elementos. Si eso es supone que la circunferencia de un círculo, entonces el método puede calcular fácilmente un radio de ese círculo. Pero ese radio es en realidad extendida por la mitad de la altura de cada Etiqueta. El punto final de ese radio de este modo coincide con el centro de cada Etiqueta. los Etiqueta está situado dentro de la AbsoluteLayout restando la mitad de la Etiqueta anchura y la altura de ese punto.

Un ángulo acumulado se utiliza tanto para encontrar el punto final de la radio para la siguiente Etiqueta y para hacer girar el Etiqueta. Debido a que el punto final de cada radio coincide con el centro de cada Etiqueta, el ángulo se incrementa sobre la base de la mitad de la anchura de la corriente Etiqueta y la mitad de la anchura de la siguiente Etiqueta.

A pesar de que la matemática es un poco complicado, el resultado vale la pena:



Este programa no establece valores predeterminados de `anchorX` y `anchorY`, así que no hay problema cambiar la orientación del teléfono en iOS.

Un reloj analógico

Uno de los programas de ejemplo clásico de una interfaz gráfica de usuario es un reloj analógico. Una vez más, Caja-Ver viene al rescate de las manecillas del reloj. Estas `BoxView` elementos deben ser girados sobre la base de las horas, minutos y segundos de la hora actual.

Primero vamos a cuidar de las matemáticas de rotación con una clase llamada `AnalogClockViewModel`, que se incluye en el `Xamarin.FormsBook.Toolkit` biblioteca:

```
espacio de nombres Xamarin.FormsBook.Toolkit
{
    clase pública AnalogClockViewModel : ViewModelBase
    {
        doble hourAngle, minuteAngle, secondAngle;

        público AnalogClockViewModel ()
        {
            UpdateLoop ();
        }

        vacío asíncrono UpdateLoop ()
        {
            mientras ( cierto )
            {
                Fecha y hora fechaHora = Fecha y hora .Ahora;
                HourAngle = 30 * (dateTime.Hour% 12) + 0,5 * dateTime.Minute;
                MinuteAngle = 6 * dateTime.Minute + 0,1 * dateTime.Second;
            }
        }
    }
}
```

```

SecondAngle = 6 * dateTIme.Second + 0,006 * dateTIme.Millisecond;

esperar Tarea .Delay (16);

}

}

pública doble HourAngle
{
    conjunto privado { SetProperty ( árbito hourAngle, valor ); }
    obtener { regreso hourAngle; }

}

pública doble MinuteAngle
{
    conjunto privado { SetProperty ( árbito minuteAngle, valor ); }
    obtener { regreso minuteAngle; }

}

pública doble SecondAngle
{
    conjunto privado { SetProperty ( árbito secondAngle, valor ); }
    obtener { regreso secondAngle; }

}
}

```

Cada una de las tres propiedades se actualiza 60 veces por segundo en un bucle de ritmo por una Task.Delay llamada. Por supuesto, el ángulo de rotación de la mano hora se basa no sólo en la hora, pero en una parte fraccionaria de esa hora disponible en el Minuto parte de Fecha y hora valor. Del mismo modo, el ángulo de la aguja de los minutos se basa en la Minuto y Segundo propiedades, y la segunda parte se basa en la Segundo y Milisecondo propiedades.

Estas tres propiedades del ViewModel se pueden unir a la Rotación propiedades de las tres manecillas del reloj analógico

Como usted sabe, algunos relojes tienen una segunda mano sin problemas de deslizamiento, mientras que la segunda parte de otros relojes mueve en las garrapatas discretas. La clase `AnalogClockViewModel` parece imponer una segunda mano suave, pero si quieres garrapatas discretas, se puede suministrar un convertidor de valores para ese propósito:

```
espacio de nombres Xamarin.FormsBook.Toolkit

{
    clase pública SecondTickConverter : IValueConverter
    {
        objeto público Convertir( objeto valor, Tipo tipo de objetivo,
        objeto parámetro, CultureInfo cultura)
        {
            regreso 6.0 * ( En t ) (( doble ) Valor / 6);
        }
    }

    objeto público ConvertBack ( objeto valor, Tipo tipo de objetivo,
        objeto parámetro, CultureInfo cultura)
    {
        regreso ( doble ) valor;
    }
}
```

}

El nombre de esta clase e incluso el código diminuta podría ser oscuro si no se sabe lo que se supone que debe hacer: la Convertir método convierte un ángulo de tipo doble que varía de 0 a 360 grados con partes fraccionarias en valores de ángulo discretas de 0, 6, 12, 18, 24, y así sucesivamente. Estos ángulos corresponden a las posiciones discretas de la segunda mano.

los **MinimalBoxViewClock** programa crea una instancia de tres BoxView elementos en su archivo XAML y se une al Rotación propiedades a las tres propiedades de AnalogClockViewModel:

```
< Pagina de contenido xmlns = "http://xamarin.com/schemas/2014/forms" 
    xmlns: x = "http://schemas.microsoft.com/winfx/2009/xaml" 
    xmlns: kit de herramientas = 
        "clr-espacio de nombres: Xamarin.FormsBook.Toolkit; montaje = Xamarin.FormsBook.Toolkit" 
    x: Class = "MinimalBoxViewClock.MinimalBoxViewClockPage" >

< ContentPage.Padding >
    < OnPlatform x: TypeArguments = " Espesor "
        iOS = " 0, 20, 0, 0 " />
</ ContentPage.Padding >

< ContentPage.Resources >
    < ResourceDictionary >
        < kit de herramientas: SecondTickConverter x: Key = " secondTick " />
    </ ResourceDictionary >
</ ContentPage.Resources >

< AbsoluteLayout Color de fondo = " Blanco "
    SizeChanged = " OnAbsoluteLayoutSizeChanged " >

    < AbsoluteLayout.BindingContext >
        < kit de herramientas: AnalogClockViewModel />
    </ AbsoluteLayout.BindingContext >

    < BoxView x: Nombre = " hourHand "
        Color = " Negro "
        Rotación = " {Binding} HourAngle " />

    < BoxView x: Nombre = " minutero "
        Color = " Negro "
        Rotación = " {Binding} MinuteAngle " />

    < BoxView x: Nombre = " segunda mano "
        Color = " Negro "
        Rotación = " {Binding SecondAngle, convertidor = () StaticResource secondTick " />
</ AbsoluteLayout >
</ Pagina de contenido >
```

El archivo de código subyacente establece los tamaños de éstos BoxView manecillas del reloj en función del tamaño de la absoluteLayout, y se establecen los lugares de manera que todas las manos apuntando hacia arriba desde el centro del reloj en la posición de las 12:00:

```

pública clase parcial MinimalBoxViewClockPage : Pagina de contenido
{
    pública MinimalBoxViewClockPage ()
    {
        InitializeComponent ();
    }

    vacío OnAbsoluteLayoutSizeChanged ( objeto remitente, EventArgs args )
    {
        AbsoluteLayout AbsoluteLayout = ( AbsoluteLayout )remitente;

        // Calcular un centro y radio para el reloj.
        Punto centro = nuevo Punto (AbsoluteLayout.Width / 2, absoluteLayout.Height / 2);
        doble radio = Mates .min (absoluteLayout.Width, absoluteLayout.Height) / 2;

        // Posición todas las manos apuntando hacia arriba desde el centro.
        AbsoluteLayout .SetLayoutBounds (hourHand,
            nuevo Rectángulo (Center.X - radio * 0.05,
                center.Y - radio * 0.6,
                radio * 0,10, el radio * 0,6));

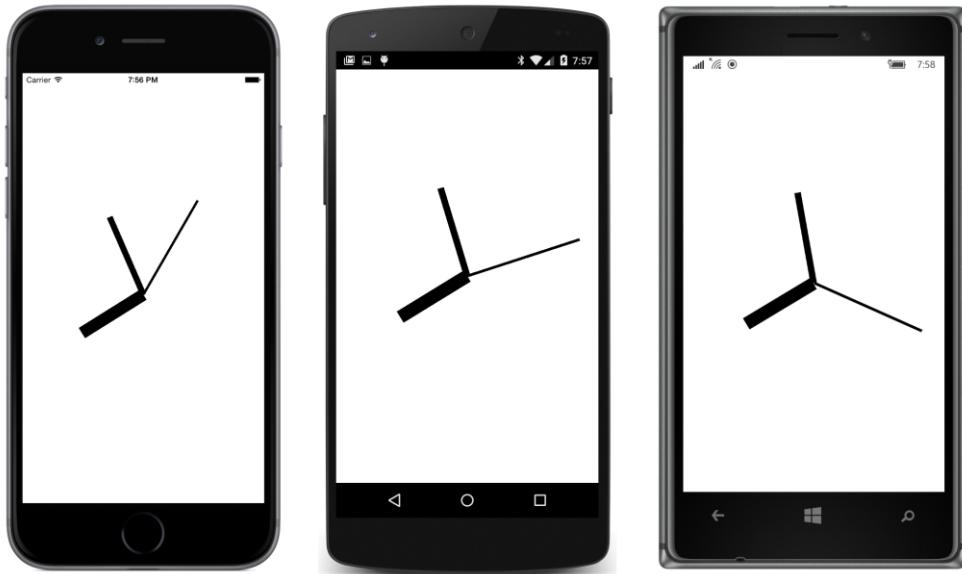
        AbsoluteLayout .SetLayoutBounds (minuteHand,
            nuevo Rectángulo (Center.X - radio * 0,025,
                center.Y - radio * 0,7,
                radio * 0,05, el radio * 0,7));

        AbsoluteLayout .SetLayoutBounds (de segunda mano,
            nuevo Rectángulo (Center.X - radio * 0,01,
                center.Y - radio * 0,9,
                radio * 0,02, el radio * 0,9));

        // Establecer el ancla a la parte inferior central de BoxView.
        hourHand.AnchorY = 1;
        minuteHand.AnchorY = 1;
        secondHand.AnchorY = 1;
    }
}

```

Por ejemplo, la aguja de las horas se da una longitud de 0,60 del radio del reloj y una anchura de 0,10 del radio del reloj. Esto significa que la posición horizontal de la esquina superior izquierda de la aguja de las horas se debe establecer en la mitad de su anchura (0,05 veces el radio) a la izquierda del centro del reloj. La posición vertical de la aguja de las horas es la altura de la mano por encima del centro del reloj. Los ajustes de anchorY asegurar que todas las rotaciones son en relación con la parte inferior central de cada mano reloj:



Por supuesto, este programa se llama **MinimalBoxViewClock** por una razón. No tiene marcas de graduación convenientes alrededor de la circunferencia, por lo que es un poco difícil de discernir el tiempo real. Además, las manecillas del reloj deben superponerse más propiamente el centro de la esfera del reloj para que al menos parecen estar unido a un alfiler o un tubo giratorio.

Ambos problemas se abordan en el nonminimal **BoxViewClock**. El archivo XAML es muy similar a **MinimalBoxViewClock**, pero el archivo de código subyacente es más extensa. Se inicia con una pequeña estructura llamada **HandParams**, que define el tamaño de cada mano con relación al radio, pero también incluye una **Compensar** valor. Esta es una fracción de la longitud total de la mano, lo que indica donde se alinea con el centro de la esfera del reloj. También se convierte en el **anchorY** valor para las rotaciones:

```

pública clase parcial BoxViewClockPage : Pagina de contenido
{
    // Estructura para almacenar información acerca de las tres manos.
    struct HandParams
    {
        público HandParams ( doble anchura, doble altura, doble compensar ) : esta ()
        {
            Anchura = ancho;
            Altura = altura;
            Offset = offset;
        }

        pública doble Ancho { conjunto privado ; obtener ; } // fracción de radio
        pública doble altura { conjunto privado ; obtener ; } // idem
        pública doble Compensar { conjunto privado ; obtener ; } // en relación con pivote central
    }

    estático solo lectura HandParams secondParams = nuevo HandParams (0,02, 1,1, 0,85);
    estático solo lectura HandParams minuteParams = nuevo HandParams (0,05, 0,8, 0,9);
}

```

```
estático solo lectura HandParams hourParams = nuevo HandParams (0,125, 0,65, 0,9);

BoxView [] Marcas de división = nuevo BoxView [60];

público BoxViewClockPage ()
{
    InitializeComponent ();

    // Crear las marcas de graduación (para ser el tamaño y posición posterior).
    para (En t i = 0; i < tickMarks.Length; i++)
    {
        las marcas de división [i] = nuevo BoxView (Color = Color .black);
        absoluteLayout.Children.Add (marcas de división [i]);
    }
}

vacío OnAbsoluteLayoutSizeChanged ( objeto remitente, EventArgs args)
{
    // Obtener el centro y el radio de la AbsoluteLayout.
    Punto centro = nuevo Punto (AbsoluteLayout.Width / 2, absoluteLayout.Height / 2);
    doble radio = 0,45 * Mates .min (absoluteLayout.Width, absoluteLayout.Height);

    // posición, el tamaño y rotar las 60 marcas de graduación.
    para (En t index = 0; índice < tickMarks.Length; índice++)
    {
        doble size = radio / (?% 5 == 0 15 index: 30);
        doble radianes = índice * 2 * Mates .PI / tickMarks.Length;
        doble x = center.X + radio * Mates .Sin (radianes) - Tamaño / 2;
        doble y = center.Y - radio * Mates .Cos (radianes) - Tamaño / 2;
        AbsoluteLayout .SetLayoutBounds (marcas de división [índice], nuevo Rectángulo (X, y, el tamaño, el tamaño));
        las marcas de división [índice] .Rotation = 180 * radianes / Mates .PI;
    }

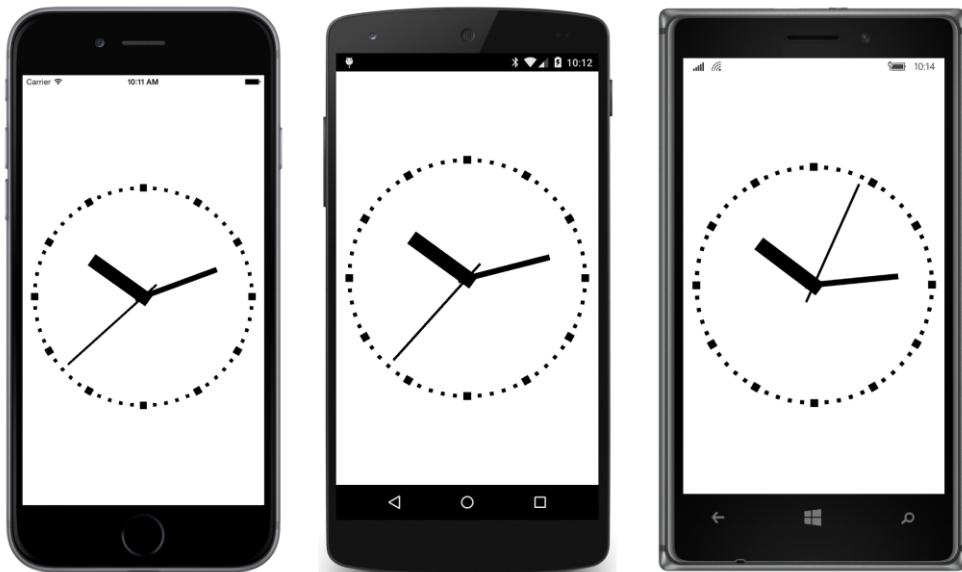
    // La posición y el tamaño de las tres manos.
    LayoutHand (de segunda mano, secondParams, centro, radio);
    LayoutHand (minuteHand, minuteParams, centro, radio);
    LayoutHand (hourHand, hourParams, centro, radio);
}

vacío LayoutHand ( BoxView boxView, HandParams handParams, Punto centrar, doble radio)
{
    doble width = handParams.Width * radio;
    doble height = handParams.Height * radio;
    doble offset = handParams.Offset;

    AbsoluteLayout .SetLayoutBounds (boxView,
        nuevo Rectángulo (Center.X - 0,5 * anchura,
            center.Y - desplazamiento * altura,
            anchura, altura));

    // Establecer la propiedad anchorY para las rotaciones.
    boxView.AnchorY = handParams.Offset;
}
```

Las marcas de la señal alrededor de la circunferencia de la esfera del reloj son también BoxView elementos, pero hay 60 de ellos con dos tamaños diferentes, y están posicionados utilizando técnicas que ya ha visto. Las imágenes son sorprendentemente bueno teniendo en cuenta la ausencia de un sistema de gráficos Xamarin.Forms:



Lo mejor de todo, en realidad se puede decir la hora.

Este reloj tiene otra característica interesante que hace que el movimiento de las manos muy fascinante. La segunda mano se desliza ni de un segundo a otro o hace saltos discretos; en cambio, tiene un movimiento más complejo. Se tira un poco hacia atrás, luego salta por delante, pero ligeramente overshooting su marca, y luego retrocede y se detiene. ¿Cómo se hace esto?

En el siguiente capítulo, verá que implementa varias Xamarin.Forms *funciones de aceleración* que pueden añadir realismo a una animación mediante el cambio de velocidad por la animación de acelerarla y frenarla-a lo largo de la animación. Tales funciones de aceleración se han vuelto bastante estándar en toda la industria de la computación, y **Xamarin.FormsBook.Toolkit** contiene un convertidor de valores que implementa una función de aceleración de la llamada *fácilidad de vuelta*:

```
espacio de nombres Xamarin.FormsBook.Toolkit
{
    clase pública SecondBackEaseConverter : IValueConverter
    {
        objeto público Convertir( objeto valor, Tipo tipo de objetivo,
            objeto parámetro, CultureInfo cultura)
        {
            En t segundos = ( En t ) ( doble ) Valor / 6; // 0, 1, 2, ... 60
            doble t = ( doble ) Valor / 6% 1; // 0 > 1
            doble v = 0; // 0 > 1

            // volver-facilidad dentro y fuera de las funciones http://robertpenner.com/easing/
        }
    }
}
```

```

Si (T <0,5)
{
    t = 2;
    v = 0,5 * T * T * ((1,7 + 1) * t - 1,7);
}
más
{
    t = 2 * (t - 0,5);
    v = 0,5 * (1 + ((t - 1) * (t - 1) * ((1,7 + 1) * (t - 1) + 1,7) + 1));
}

regreso 6 * (segundo + v);
}

objeto público ConvertBack (objeto valor, Tipo tipo de objetivo,
                                objeto parámetro, CultureInfo cultura)
{
    regreso (doble)valor;
}
}
}
}

```

Este convertidor se hace referencia en el **BoxViewClock** archivo XAML:

```

< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    xmlns: kit de herramientas =
        " cl-espacio de nombres: Xamarin.FormsBook.Toolkit; montaje = Xamarin.FormsBook.Toolkit "
    x: Class = " BoxViewClock.BoxViewClockPage " >

< ContentPage.Padding >
    < OnPlatform x: TypeArguments = " Espesor "
        iOS = " 0, 20, 0, 0 " />
</ ContentPage.Padding >

< ContentPage.Resources >
    < ResourceDictionary >
        < kit de herramientas: SecondBackEaseConverter x: Key = " secondBackEase " />
    </ ResourceDictionary >
</ ContentPage.Resources >

< AbsoluteLayout x: Nombre = " AbsoluteLayout "
    Color de fondo = " Blanco "
    SizeChanged = " OnAbsoluteLayoutSizeChanged " >

    < AbsoluteLayout.BindingContext >
        < kit de herramientas: AnalogClockViewModel />
    </ AbsoluteLayout.BindingContext >

    < BoxView x: Nombre = " hourHand "
        Color = " Negro "
        Rotación = " {Binding} HourAngle " />

    < BoxView x: Nombre = " minutero "
        Color = " Negro "
        Rotación = " {Binding} MinuteAngle " />

```

```
< BoxView x: Nombre = "segunda mano "
    Color = "Negro "
    Rotación = "(Binding SecondAngle, convertidor = {} ) StaticResource secondBackEase " />
</ AbsoluteLayout >
</ Pagina de contenido >
```

Usted verá más funciones de aceleración en el siguiente capítulo.

deslizadores verticales?

Pueden ciertos puntos de vista pueden girar y todavía funcionan como deberían? Más específicamente, puede la horizontal normal de deslizador elementos de Xamarin.Forms pueden girar para convertirse en controles deslizantes verticales?

Vamos a intentarlo. Los **VerticalSliders** programa contiene tres deslizadores en una StackLayout, y el Apilar-Diseño sí se gira 90 grados a la izquierda:

```
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = "VerticalSliders.VerticalSlidersPage " >

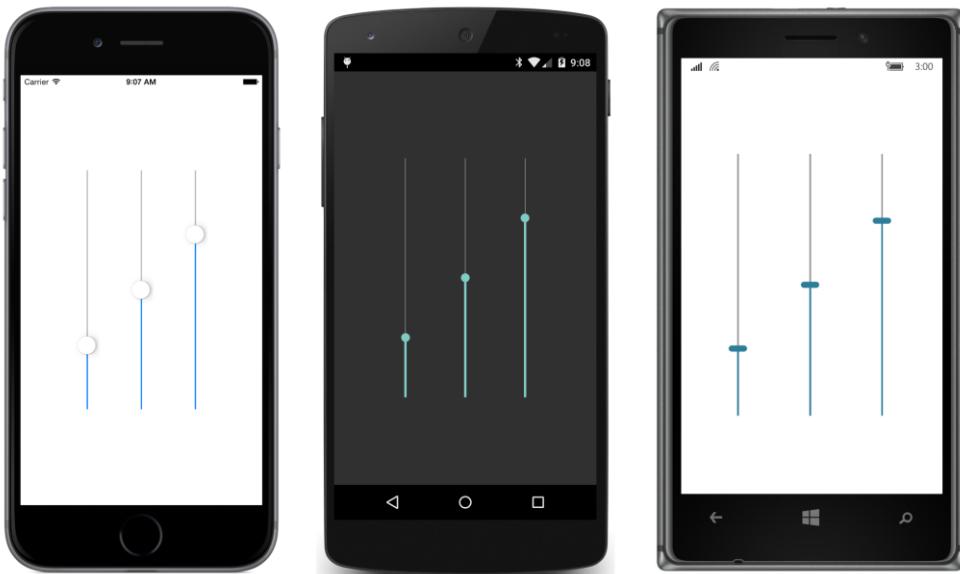
    < StackLayout VerticalOptions = "Centrar "
        Espaciado = "50 "
        Rotación = "-90 " >

        < deslizador Valor = "0.25 " />

        < deslizador Valor = "0.5 " />

        < deslizador Valor = "0.75 " />
    </ StackLayout >
</ Pagina de contenido >
```

Efectivamente, los tres deslizadores están orientados verticalmente:



Y trabajan! Puede manipular las barras de desplazamiento verticales, justo como si hubieran sido diseñado para tal fin. Los Mínimo valor corresponde a una posición del pulgar en la parte inferior, y el Máximo valor corresponde a la parte superior.

Sin embargo, el sistema de diseño Xamarin.Forms es completamente inconsciente de las nuevas ubicaciones de las barras de desplazamiento. Por ejemplo, si a su vez el teléfono al modo horizontal, los controles deslizantes cambian de tamaño para el ancho de la pantalla vertical y son demasiado grandes para ser girado a una posición vertical. Tendrá que pasar algún esfuerzo extra para conseguir deslizadores giradas posicionado y dimensionado de forma inteligente.

Pero funciona.

rotaciones 3D-ish

A pesar de que las pantallas de ordenador son planas y de dos dimensiones, es posible dibujar objetos visuales en estas pantallas que dan la apariencia de una tercera dimensión. Al principio de este capítulo le vio algunos efectos de texto que dan el toque de una tercera dimensión, y Xamarin.Forms soporta dos rotaciones adicionales, llamados `rotationX` y `rotationY`, que también parecen romper a través de la planitud de dos dimensiones inherentes de la pantalla.

Cuando se trata de gráficos en 3D, es conveniente pensar en la pantalla como parte de un sistema de coordenadas 3D. El eje X es horizontal y el eje Y es vertical, como de costumbre. Pero también hay un eje Z implícita de que es ortogonal a la pantalla. Este eje Z sobresale de la pantalla y se extiende a través de la parte posterior de la pantalla.

En el espacio 2D, la rotación se produce alrededor de un punto. En el espacio 3D, la rotación se produce alrededor de un eje. La `rotationX` propiedad es la rotación alrededor del eje X. La parte superior e inferior de un objeto visual parecen moverse hacia el observador o lejos del espectador. Similar, `rotationY` es la rotación alrededor del eje Y. Los lados izquierdo y derecho de un objeto visual parecen moverse hacia el observador o lejos del espectador. Por extensión, el básico `Rotación` propiedad es la rotación alrededor del eje Z. Por consistencia, la `Rotación` Probablemente la propiedad debe ser nombrado `rotationZ`, pero que podría confundir a las personas que están pensando sólo en dos dimensiones.

los **ThreeDeeRotationDemo** programa le permite experimentar con combinaciones de `rotationX`, `rotationY`, y `Rotación`, así como explorar cómo la `anchorX` y `anchorY` afectar a estas dos propiedades de rotación adicionales:

```
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns:x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " ThreeDeeRotationDemo.ThreeDeeRotationDemoPage " >

< StackLayout Relleno = " 20, 10 " >
    < Marco x: Nombre = " marco " >
        HorizontalOptions = " Centrar "
        VerticalOptions = " CenterAndExpand "
        OutlineColor = " Acento " >

        < Etiqueta Texto = " TEXTO " >
            Tamaño de fuente = " 72 " />
        </ Marco >

        < deslizador x: Nombre = " rotationXSlider " >
            Máximo = " 360 "
            Valor = " {Binding Fuente = {x: Marco de Referencia},
                        Path = rotationX} " />

        < Etiqueta Texto = " {Binding Fuente = {x: rotationXSlider Referencia},
                        Path = Valor,
                        StringFormat = 'rotationX = {0: F0}'} "
            HorizontalTextAlignment = " Centrar " />

        < deslizador x: Nombre = " rotationYSlider " >
            Máximo = " 360 "
            Valor = " {Binding Fuente = {x: Marco de Referencia},
                        Path = rotationY} " />

        < Etiqueta Texto = " {Binding Fuente = {x: rotationYSlider Referencia},
                        Path = Valor,
                        StringFormat = 'rotationY = {0: F0}'} "
            HorizontalTextAlignment = " Centrar " />

        < deslizador x: Nombre = " rotationZSlider " >
            Máximo = " 360 "
            Valor = " {Binding Fuente = {x: Marco de Referencia},
                        Path = Rotación} " />

        < Etiqueta Texto = " {Binding Fuente = {x: rotationZSlider Referencia},
                        Path = Valor,
```

```
StringFormat = 'de rotación (Z) = {0: F0}' "
HorizontalTextAlignment = "Centrar" />

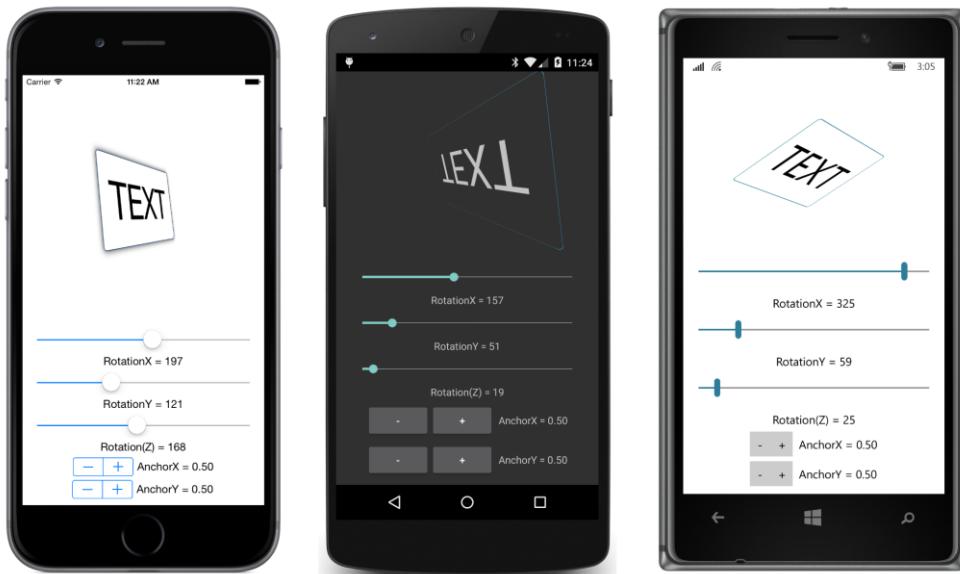
< StackLayout Orientación = "Horizontal" 
    HorizontalOptions = "Centrar" >
        < paso a paso x: Nombre = "anchorXStepper" 
            Mínimo = "-1"
            Máximo = "2"
            Incremento = "0.25"
            Valor = "{Binding Fuente = {x: Marco de Referencia},
                        Path = anchorX}" />

        < Etiqueta Texto = "{Binding Fuente = {x: anchorXStepper Referencia}},
                        Path = Valor,
                        StringFormat = 'anchorX = {0: F2}' "
            VerticalOptions = "Centrar" />
    </ StackLayout >

    < StackLayout Orientación = "Horizontal" 
        HorizontalOptions = "Centrar" >
            < paso a paso x: Nombre = "anchorYStepper" 
                Mínimo = "-1"
                Máximo = "2"
                Incremento = "0.25"
                Valor = "{Binding Fuente = {x: Marco de Referencia},
                            Path = anchorY}" />

            < Etiqueta Texto = "{Binding Fuente = {x: anchorYStepper Referencia}},
                            Path = Valor,
                            StringFormat = 'anchorY = {0: F2}' "
                VerticalOptions = "Centrar" />
        </ StackLayout >
    </ StackLayout >
</ Página de contenido >
```

He aquí un ejemplo de pantalla que muestra las combinaciones de las tres rotaciones:



Usted descubrirá que la `anchorY` propiedad afecta `rotationX` pero no `rotationY`. Para el valor por defecto `anchorY` valor de 0,5, `rotationX` provoca la rotación que se produzca alrededor del centro horizontal del objeto visual. Cuando se establece `anchorY` a 0, la rotación es de alrededor de la parte superior del objeto, y para un valor de 1, la rotación es de alrededor de la parte inferior.

Del mismo modo, la `anchorX` propiedad afecta `rotationY` pero no `rotationX`. Un `anchorX` valor de 0 causa `rotationY` para rotar el objeto visual alrededor de su borde izquierdo, mientras que un valor de 1 provoca la rotación alrededor del borde derecho.

Las direcciones de rotación son consistentes entre las tres plataformas, sino que se describen mejor en relación con las convenciones de sistemas de coordenadas en 3D:

Se podría pensar que hay muchas maneras de organizar ortogonales X, Y, y Z. Por ejemplo, los valores crecientes de X podrían aumentar correspondiente a hacia la izquierda o movimiento hacia la derecha en el eje X, y los valores crecientes de Y pueden corresponder con arriba o hacia abajo el movimiento en el eje Y. Sin embargo, muchas de estas variaciones se hacen equivalentes cuando los ejes son vistos desde diferentes direcciones. En realidad, sólo hay dos diferentes maneras de organizar X, Y, y Z. Estas dos formas se conocen como *mano derecha* y *mano izquierda* sistemas coordinados.

El sistema de coordenadas 3D implicado por los tres Rotación propiedades en Xamarin.Forms es zurdo: Si apunta el dedo índice de la mano izquierda en la dirección de aumentar coordenadas X (que está a la derecha), y el dedo medio en la dirección de aumentar coordenadas Y (que está abajo), entonces su pulgar apunta en la dirección de aumentar coordenadas Z, que están saliendo de la pantalla.

Su mano izquierda también se puede utilizar para predecir la dirección de rotación: Para la rotación alrededor de un eje en particular, primer punto de su dedo pulgar en la dirección de aumentar los valores en ese eje. Para la rotación alrededor de la X

eje, coloque el pulgar izquierdo derecha. Para la rotación alrededor del eje Y, apuntar su dedo pulgar izquierdo hacia abajo. Para la rotación alrededor del eje Z, apunta su dedo pulgar izquierdo que sale de la pantalla. El rizo de los otros dedos de la mano izquierda indica el sentido de rotación de los ángulos positivos.

En resumen:

- Para aumentar los ángulos de rotationX, la parte superior va y la parte inferior sale.
- Para aumentar los ángulos de rotationY, el lado derecho se remonta y el lado izquierdo sale.
- Para aumentar los ángulos de Rotación, la rotación es en sentido horario.

Aparte de estas convenciones, rotationX y rotationY no presentan mucha coherencia visual entre las tres plataformas. Aunque las tres plataformas implementan perspectiva, es decir, la parte del objeto aparentemente más cercano a la vista es mayor que la parte del objeto más lejos-la cantidad de perspectiva que verá es específica de la plataforma. No hay AnchorZ propiedad que podría permitir afinar estas imágenes.

Pero lo que es quizás más evidente es que estos diversos Rotación propiedades serían muy divertido para animar.

capítulo 22

Animación

La animación es la vida, acción, vitalidad, y en los equipos que tratan de imitar esas cualidades a pesar de estar restringido a la manipulación de diminutos píxeles en una pantalla plana.

animación por ordenador por lo general se refiere a cualquier tipo de cambio visual dinámica. UN Botón que sólo aparece en una página no es la animación. Sin embargo, una Botón que se desvanece a la vista, o se mueve en su lugar, o crece en tamaño, desde un punto-que es la animación. Muy a menudo, los elementos visuales responden a la entrada del usuario con un cambio en el aspecto, tal como una Botón flash, una paso a paso incremento, o una Vista de la lista voluta. Eso, también, es la animación.

A veces es deseable para una aplicación para ir más allá de las animaciones automáticas y convencionales y añadir su propio. Eso es lo que este capítulo se trata.

Usted comenzó a ver algo de esto en el capítulo anterior. Ya viste cómo configurar transforma en elementos visuales y luego usar el temporizador o Task.Delay para animar ellos. Xamarin.Forms también incluye su propia infraestructura de animación que existe en tres niveles de interfaces de programación correspondiente a las clases ViewExtensions, Animación, y AnimationExtensions. Este sistema de animación es lo suficientemente versátil como para trabajos complejos, pero excepcionalmente fácil para trabajos sencillos. Este capítulo comienza con la clase fácil de alto nivel (ViewExtensions) y luego profundiza hasta niveles inferiores a los más versátiles.

Las clases de animación Xamarin.Forms se utilizan generalmente para apuntar propiedades de los elementos visuales. Una animación típica cambia progresivamente una propiedad de un valor a otro valor en un período de tiempo. Las propiedades que son el objetivo de las animaciones deben ser respaldados por las propiedades enlazables. Este no es un requisito, pero las propiedades enlazables son generalmente diseñados para responder a los cambios dinámicos a través de la implementación de un controlador de propiedad cambiado. No es bueno para animar una propiedad de un objeto si el objeto no tiene ni siquiera se da cuenta de que la propiedad está siendo cambiado!

No hay ninguna interfaz XAML para el sistema de animación Xamarin.Forms. En consecuencia, todas las animaciones en este capítulo se realizan en el código. Sin embargo, como se verá en el siguiente capítulo, se puede encapsular animaciones en clases llamadas *las acciones de activación y comportamientos*, y luego hacer referencia a ellos a partir de archivos XAML. Disparadores y comportamientos son generalmente la forma más fácil (y la forma recomendada) para incorporar animaciones dentro de las aplicaciones MVVM.

Explorar animaciones básicas

Vamos a bucear con un pequeño programa llamado **AnimationTryout**. El archivo XAML contiene nada más que una centrada Botón:

```
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns:x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " AnimationTryout.AnimationTryoutPage " >

    < Botón x: Nombre = " botón "
        Texto = " Puntee en Yo! "
        Tamaño de fuente = " Grande "
        HorizontalOptions = " Centrar "
        VerticalOptions = " Centrar "
        hecho clic = " OnButtonClicked " />

</ Pagina de contenido >
```

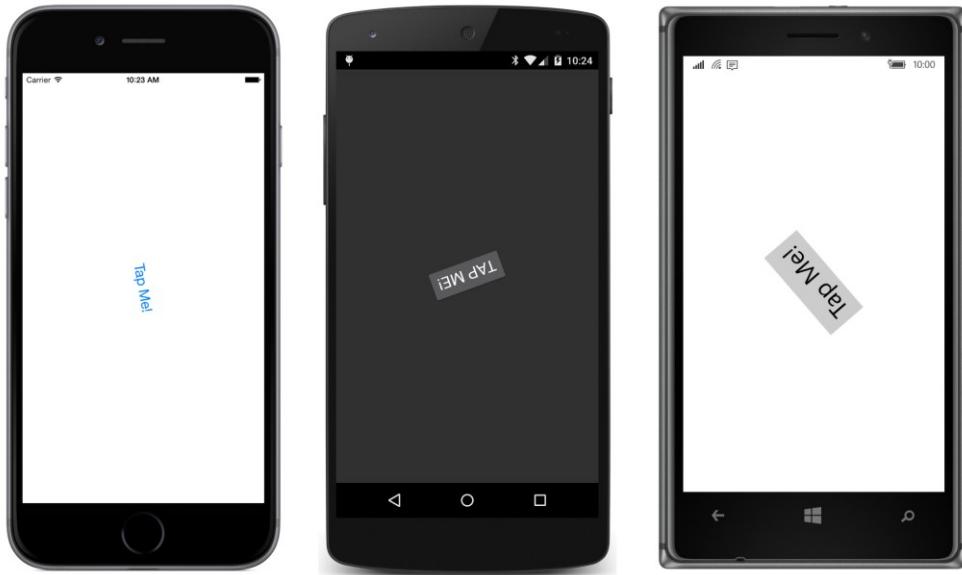
Para este ejercicio, vamos a ignorar la función esencial que la real Botón presumiblemente lleva a cabo dentro de la aplicación. Además de querer el botón para llevar a cabo esa función, supongamos que desea girar alrededor de un círculo cuando el usuario toca la misma. Los hecho clic manejador en el archivo de código subyacente puede hacer que al llamar a un método llamado `RotateTo` con un argumento de 360 para el número de grados para girar:

```
público clase parcial AnimationTryoutPage : Pagina de contenido
{
    público AnimationTryoutPage ()
    {
        InitializeComponent ();
    }

    vacío OnButtonClicked ( objeto remitente, EventArgs args)
    {
        button.RotateTo (360);
    }
}
```

Los `RotateTo` método es una animación que se dirige a la Rotación propiedad de Botón. Sin embargo, el `RotateTo` método no está definido en el `VisualElement` clase como el Rotación propiedad. Es, en cambio, un método de extensión se define en la `ViewExtensions` clase.

Cuando se ejecuta este programa y pulsa el botón, la `RotateTo` Método anima el botón para girar alrededor de un círculo completo de 360 grados. Aquí está en curso:



El viaje completo dura 250 milisegundos (un cuarto de segundo), que es la duración predeterminada de esta `RotateTo` animación.

Sin embargo, este programa tiene un defecto. Después de haber visto el botón de girar alrededor, intente tocar de nuevo. Que no gire.

Ese defecto del programa revela un poco acerca de lo que está pasando internamente: En la primera llamada a `OnButtonClicked`, el `RotateTo` método obtiene la corriente `Rotación` propiedad, que es 0, y luego define una animación de la `Rotación` la propiedad de ese valor para el argumento de `RotateTo`, cual es

360. Cuando la animación llega a la conclusión después de un cuarto de segundo, el `Rotación` se deja la propiedad en 360.

La próxima vez que se pulsa el botón, el valor actual es de 360 y el argumento de `RotateTo` es también 360. Internamente, la animación se sigue produciendo, pero la `Rotación` la propiedad no se mueve. De que está pegada a 360.

Ajuste de la duración de la animación

He aquí una pequeña variación de la hecho clic manejador de `AnimationTryout`. No se soluciona el problema con varios grifos de la Botón, pero se extiende la animación de dos segundos para que pueda disfrutar de la animación más largo. La duración se especifica en milisegundos como el segundo argumento de `RotateTo`. Ese segundo argumento es opcional y tiene un valor por defecto de 250:

```
vacio OnButtonClicked ( objeto remitente, EventArgs args)
{
    button.RotateTo (360, 2000);
}
```

Con esta variación, intente sacudir el Botón y luego tocando de nuevo varias veces, ya que está girando.

Usted descubrirá que los repetidos toques del botón no envíe el Rotación espalda propiedad en cero. En cambio, la animación anterior se cancela y se inicia una nueva animación. Pero esta nueva animación comienza en cualquiera que sea el Rotación la propiedad pasa a ser en el momento de la toma. Cada nueva animación todavía tiene una duración de 2 segundos, pero la corriente Rotación la propiedad está más cerca del valor final de 360 grados, por lo que cada nueva animación parece ser más lenta que la anterior. Despues de la Rotación propiedad finalmente llega a 360, sin embargo, más grifos no hacen nada.

animaciones relativas

Una solución al problema de los grifos posteriores es utilizar RelRotateTo ("Girar en relación con"), que obtiene la corriente Rotación propiedad para el inicio de la animación y luego añade a su argumento de que el valor para el final de la animación. He aquí un ejemplo:

```
vacio OnButtonClicked ( objeto remitente, EventArgs args)
{
    button.RelRotateTo (90, 1000);
}
```

Cada caja de derivación se inicia una animación que gira el botón otros 90 grados a lo largo de un segundo. Si le sucede a pulsar el botón, mientras que una animación está en curso, una nueva animación se inicia desde esa posición, por lo que podría terminar en una posición que no es un incremento de 90 grados. No hay ningún cambio en la velocidad con múltiples grifos porque la animación siempre va a un ritmo de 90 grados por segundo.

Ambos RotateTo y RelRotateTo tener una estructura subyacente común. Durante el transcurso de la animación, se calcula un valor-a menudo llamado *t* (por el tiempo) o, a veces, *Progreso*. Este valor se basa en el tiempo transcurrido y la duración de la animación:

$$\bullet = \frac{\text{-----}}{\text{-----}}$$

Valores de *t* variar desde 0 al comienzo de la animación a 1 al final de la animación. La animación también está definida por dos valores (a menudo los valores de una propiedad), uno para el inicio de la animación y otro para el final. Estos a menudo se denominan *comienzo* y *fin* valores, o *de* y *a* valores. La animación calcula un valor de entre *de* y *a* basa en una simple fórmula de interpolación:

$$\text{.....} = \text{.....} + \bullet \cdot (\text{.....} - \text{.....})$$

Cuando *t* es igual a 0, *valores* igual *fromValue* y cuando *t* es igual a 1, *valores* igual *valor*.

Ambos RotateTo y RelRotateTo obtener *fromValue* a partir del valor actual de la Rotación propiedad en el momento de llamar al método. RotateTo conjuntos *valor* igual a su argumento, mientras RelRotateTo conjuntos *valor* igual a *fromValue* además de su argumento.

En espera de animaciones

Otra forma de solucionar el problema con las líneas siguientes es inicializar el Rotación establecimiento antes de la llamada a RotateTo:

```
vacio OnButtonClicked ( objeto remitente, EventArgs args)
{
    button.Rotation = 0;
    button.RotateTo (360, 2000);
}
```

Ahora puede pulsar el Botón de nuevo después de que se detuvo y se comenzará la animación desde el principio. golpecitos repetidos mientras que el Botón está girando también se comportan de manera diferente: Ellos volver a empezar desde 0 grados.

Curiosamente, esta ligera variación en el código hace *no* permitir grifos posteriores:

```
vacio OnButtonClicked ( objeto remitente, EventArgs args)
{
    button.RotateTo (360, 2000);
    button.Rotation = 0;
}
```

Esta versión se comporta igual que la versión con sólo el RotateTo método. Parece como si el establecimiento de la Rotación propiedad a 0 después de esa llamada no hace nada.

¿Por qué no funciona? No funciona porque el RotateTo método es asíncrono. El método devuelve rápidamente después de iniciar la animación, pero la animación en sí ocurre en el fondo. Ajuste de la Rotación propiedad a 0 en el momento de la RotateTo devuelve el método no tiene ningún efecto aparente debido a que la configuración es reemplazada rápidamente por el fondo RotateTo animación.

Debido a que el método es asíncrona, RotateTo devuelve una Tarea objeto más específicamente, una

Tarea <bool> OBJETO DE y eso significa que se puede llamar Continua con para especificar una función de devolución de llamada que se invoca cuando termina la animación. La devolución de llamada puede entonces establecer la Rotación propiedad de recuperación a 0 después de la animación se ha completado:

```
vacio OnButtonClicked ( objeto remitente, EventArgs args)
{
    button.RotateTo (360, 2000) .ContinueWith ((tarea) =>
    {
        button.Rotation = 0;
    });
}
```

los tarea objeto pasado a Continua con es de tipo Tarea <bool>, y el Continua con devolución de llamada puede utilizar la Resultado propiedad para obtener ese valor booleano. El valor es cierto si la animación fue cancelado y falso si funcionara hasta su finalización. Se puede confirmar fácilmente esta mostrando el valor de retorno utilizando una Debug.WriteLine llamar y mirando a los resultados en el Salida ventana de Visual Studio o Xamarin Estudio:

```
vacio OnButtonClicked ( objeto remitente, EventArgs args)
{
    button.RotateTo (360, 2000) .ContinueWith ((tarea) =>
    {
        Diagnostico del sistema. Depurar .Línea de escritura( "¿Cancelado? " + Task.Result);
        button.Rotation = 0;
    });
}
```

```
});  
}
```

Si toca el Botón mientras que se está animado, verá cierto Los valores devueltos. Cada nueva llamada a RotateTo cancela la animación anterior. Si deja que la animación se termina de ejecutar, verá una falso valor devuelto.

Es más probable que va a utilizar esperar con el RotateTo método de Continua con:

```
vacio asíncrono OnButtonClicked ( objeto remitente, EventArgs args)  
{  
    bool wasCancelled = esperar button.RotateTo (360, 2000);  
    button.Rotation = 0;  
}
```

O bien, si no se preocupan por el valor de retorno:

```
vacio asíncrono OnButtonClicked ( objeto remitente, EventArgs args)  
{  
    esperar button.RotateTo (360, 2000);  
    button.Rotation = 0;  
}
```

Observe la asíncrono modificador en el controlador, que se requiere para cualquier método que contiene esperar operadores.

Si ha utilizado otros sistemas de animación, es muy probable que se le requirió para definir un método de devolución de llamada si quería la aplicación que se le notifique cuando se ha completado una animación. Con esperar, determinar cuándo se ha completado, tal vez una animación para ejecutar algún otro código-convertido en algo trivial. En este ejemplo particular, el código que se ejecuta es bastante simple, pero por supuesto que podría ser más compleja.

A veces, usted desea que sus animaciones simplemente ejecuten por completo en el fondo, en cuyo caso no es necesario el uso de esperar con ellos y, a veces usted querrá hacer algo cuando la animación se ha completado. Pero ten cuidado: si el Botón También está provocando una cierta función aplicación real, puede que no quiera esperar hasta que termine la animación antes de llevar a cabo que.

RotateTo y RelRotateTo son dos de los varios métodos similares se define en la ViewExtensions clase. Otros que verá en este capítulo incluyen ScaleTo, TranslateTo, fadeTo, y LayoutTo. Todos vuelven Tarea <bool> objetos- falso si la animación terminada sin interrupción y cierto si fue cancelado.

Su aplicación puede cancelar una o más de estas animaciones con una llamada al método estático ViewExtensions.CancelAnimations. A diferencia de todos los otros métodos en ViewExtensions, esto no es un método de extensión. Es necesario llamarlo así:

```
ViewExtensions .CancelAnimations (botón);
```

Eso va a cancelar de inmediato todas las animaciones iniciados por los métodos de extensión en el ViewExtensions clase que se están ejecutando actualmente en el botón objeto.

Utilizando esperar es particularmente útil para el apilamiento animaciones secuenciales:

```
vacio asincrono OnButtonClicked ( objeto remitente, EventArgs args)
{
    esperar button.RotateTo (90, 250);
    esperar button.RotateTo (-90, 500);
    esperar button.RotateTo (0, 250);
}
```

La animación total definido aquí requiere un segundo. los Botón columpios 90 grados hacia la derecha en el primer trimestre segundos, a continuación, 180 grados hacia la izquierda en el siguiente medio segundo, y luego 90 grados hacia la derecha para acabar a 0 grados de nuevo. Necesitas esperar en los dos primeros para que sean secuenciales, pero que no lo necesitan en la tercera, si no hay nada más para ejecutar en el hecho clic manejador después de la tercera animación se ha completado.

Una animación compuesto como este es a menudo conocido como *animación fotograma clave*. Está especificando una serie de ángulos de rotación y tiempos, y la animación en general es la interpolación entre ellos. En la mayoría de los sistemas de animación, animaciones de fotogramas clave son a menudo más difícil de usar que las animaciones sencillas. Pero con esperar, animaciones de fotogramas clave se convierten en triviales.

El valor de retorno de Tarea <bool> no indica necesariamente que la animación se está ejecutando en un subproceso secundario. De hecho, al menos parte de la animación -la parte que realmente establece el Rotación propiedad debe ejecutar en el hilo de interfaz de usuario. En teoría, es posible que toda la animación se ejecute en el hilo de interfaz de usuario. Como se vio en el capítulo anterior, las animaciones que cree con Device.StartTimer o Task.Delay ejecutar enteramente en el hilo de interfaz de usuario, aunque el mecanismo temporizador subyacente podría implicar un hilo secundario.

Verá más adelante en este capítulo cómo un método de animación todavía puede devolver una Tarea objeto, sino ejecutar enteramente en el hilo de interfaz de usuario. Esta técnica permite que el código para utilizar temporizadores para la estimulación animaciones, pero todavía proporcionan una forma estructurada Tarea- notificación basada en el código se ha completado.

animaciones compuestas

Usted puede mezclar esperado y llamadas nonawaited para crear animaciones compuestas. Por ejemplo, supongamos que desea el botón para girar alrededor de 360 grados al mismo tiempo que se expande en los contratos de tamaño y luego.

los ViewExtensions clase define un nombre de método scaleTo que anima a los Escala al igual que la propiedad RotateTo anima el Girar propiedad. La expansión y contracción de la Botón tamaño requiere dos animaciones secuenciales, pero éstos deben producirse al mismo tiempo que la rotación, que sólo requiere una llamada. Por esa razón, el RotateTo llamada puede ejecutar sin una esperar, y mientras que la animación se ejecuta en segundo plano, el método puede hacer dos llamadas secuenciales a ScaleTo.

Pruebe esto en AnimationTryout:

```
vacio asincrono OnButtonClicked ( objeto remitente, EventArgs args)
{
    button.Rotation = 0;
    button.RotateTo (360, 2000);
    esperar button.ScaleTo (5, 1000);
```

```
    esperar button.ScaleTo (1, 1000);
}
```

Las duraciones se hacen algo más largo de lo que serían normalmente de manera que se puede ver lo que está pasando. Los RotateTo método devuelve inmediatamente, y el primer scaleTo animación comienza en ese momento. Pero eso esperar operador en el primer scaleTo retrasa la llamada de la segunda scaleTo hasta que la primera scaleTo ha completado. En ese momento, la RotateTo animación está a medio terminar y la

Botón ha girado 180 grados. Durante los próximos 1.000 milisegundos, que RotateTo se completa en aproximadamente el mismo tiempo que el segundo scaleTo animación completa.

Aquí está la Botón como se trata de hacer su camino a través de la animación:



Porque el OnButtonClicked método se encuentra en posición con el asíncrono palabra clave y el primer RotateTo no tiene una esperar operador, obtendrá un mensaje de advertencia del compilador que afirma: "Debido a que esta llamada no se esperaba, la ejecución del método actual continúa antes de que se complete la llamada. Considerar la aplicación del operador espera 'al resultado de la llamada".

Si prefiere no ver que el mensaje de advertencia, puede apagarlo con un # Pragma afirmación de que deshabilita la advertencia de que en particular:

```
# Pragma deactivate la advertencia 4014
```

Se podría colocar esta declaración en la parte superior de su archivo de código fuente para desactivar las advertencias en todo el archivo. O se puede colocar antes de la llamada delincuencia y volver a activar las advertencias después de la llamada mediante el uso de:

```
# Pragma advirtiendo restaurar 4014
```

Task.WhenAll y Task.WhenAny

Otra opción disponible es de gran alcance que le permite combinar animaciones de una manera muy estructurada sin preocuparse por las **advertencias del compilador**. La **estática Task.WhenAll y Task.WhenAny métodos de la Tarea clase** tienen la intención de ejecutar varios métodos asíncronos simultáneamente. Cada uno de estos métodos puede aceptar una matriz o otra colección de varios argumentos, cada uno de los cuales es un método que devuelve una **Tarea objeto**. Los **Task.WhenAll y Task.WhenAny métodos también vuelven Tarea objetos**. Los **Cuando todo** método se completa cuando todos los métodos en su colección han completado. Los **WhenAny** método se completa cuando cualquier método en su colección completa su ejecución, mientras que los otros métodos en el **WhenAny** colección continúe funcionando.

Cuidado con: El **Tarea clase** también incluye métodos estáticos nombradas **WaitAll** y **WaitAny**. Usted no desea utilizar esos métodos. Bloquean el hilo de interfaz de usuario hasta que la tarea o tareas se han completado.

Porque el **Task.WhenAll** y **Task.WhenAny** métodos propios de retorno **Tarea objetos**, se pueden utilizar esperar con ellos. He aquí una manera de implementar la animación compuesta mostrado anteriormente sin ningún tipo de advertencias del compilador: La **Task.WhenAny** llamadas contiene dos tareas, la primera de las cuales tiene una duración de dos segundos y el segundo tiene una duración de un segundo. Cuando esa segunda tarea se completa, el **Task.WhenAny** llamada también completa. Los **RotateTo** método todavía está en marcha, pero ahora el segundo **scaleTo** método puede comenzar:

```
vacio asíncrono OnButtonClicked ( objeto remitente, EventArgs args)
{
    button.Rotation = 0;

    esperar Tarea .WhenAny < bool >
    (
        button.RotateTo (360, 2000),
        button.ScaleTo (5, 1000)
    );
    esperar button.ScaleTo (1, 1000);
}
```

También puedes usar **Task.Delay** con estos métodos para introducir pequeños retrasos en la animación compuesta.

Rotación y anclajes

Los **anchorX** y **anchorY** propiedades establecen el centro de la escala o la rotación para el Escala y Lista-
ción propiedades, por lo que también afectan a la **scaleTo** y **RotateTo** animaciones.

Los **CircleButton** hace girar un programa de Botón en un círculo, pero no como el código que has visto anteriormente. Este programa hace girar una Botón alrededor del centro de la pantalla, y para ello se requiere **anchorX** y **AnchorY**.

El archivo XAML pone el Botón en una **AbsoluteLayout**:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns:x = " http://schemas.microsoft.com/winfx/2009/xaml "
```

```

x: Class = " CircleButton.CircleButtonPage " >
< ContentPage.Padding >
< OnPlatform x: TypeArguments = " Espesor "
    iOS = " 0, 20, 0, 0 " />
</ ContentPage.Padding >

< AbsoluteLayout x: Nombre = " AbsoluteLayout "
    SizeChanged = " OnSizeChanged " >
< Botón x: Nombre = " botón "
    Texto = " Puntee en Yo! "
    Tamaño de fuente = " Grande "
    SizeChanged = " OnSizeChanged "
    hecho clic = " OnButtonClicked " />
</ AbsoluteLayout >
</ Pagina de contenido >

```

La única razón por este programa utiliza una **AbsoluteLayout** Para el Botón es colocar el Botón precisamente en un lugar determinado en la pantalla. El archivo XAML establece la misma **SizeChanged** manejador tanto en el **AbsoluteLayout** y el Botón. Ese controlador de eventos guarda el centro de la Absolut-

Diseño como el Punto campo denominado centrar y también ahora la distancia desde ese centro hacia el borde más cercano como el radio campo:

```

pública clase parcial CircleButtonPage : Pagina de contenido
{
    Punto centrar;
    doble radio;

    pública CircleButtonPage ()
    {
        InitializeComponent ();
    }

    vacío OnSizeChanged ( objeto remitente, EventArgs args)
    {
        centro = nuevo Punto (AbsoluteLayout.Width / 2, absoluteLayout.Height / 2);
        radio = Mathes .min (absoluteLayout.Width, absoluteLayout.Height) / 2;
        AbsoluteLayout .Set.layoutBounds (botón,
            nuevo Rectángulo (Center.X - button.Width / 2, center.Y - radio,
                AbsoluteLayout .Tamaño automático,
                AbsoluteLayout .Tamaño automático));
    }
    ...
}

```

los **OnSizeChanged** controlador concluye mediante el posicionamiento de la Botón en el centro horizontal de la página, pero con su borde superior una distancia de radio por encima del centro de la **AbsoluteLayout**:

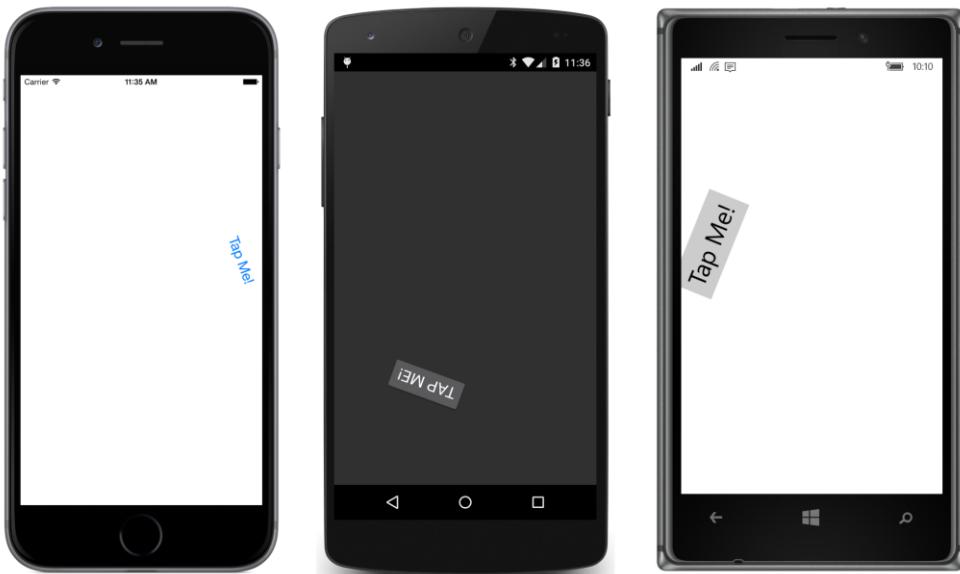


Recordemos que el `anchorX` y `anchorY` propiedades se deben establecer en números que son en relación con la anchura y la altura de la Botón. Un `anchorX` valor de 0 se refiere al borde izquierdo de la Botón y un valor de 1 se refiere a la borde derecho. Del mismo modo, una `anchorY` valor de 0 se refiere a la parte superior de la Botón y un valor de 1 se refiere a la parte inferior.

Para girar este Botón alrededor del punto de guardado como centro, `anchorX` y `anchorY` debe establecerse en valores basados en la centrar punto. El centro de la Botón está directamente encima del centro de la página, por lo que el valor predeterminado 0,5 de `anchorX` está bien. `anchorY`, Sin embargo, necesita un valor de la parte superior de la Perotonelada al punto central, sino en unidades de altura del botón:

```
    público clase parcial CircleButtonPage : Pagina de contenido
    {
        ...
        vacío asíncrono OnButtonClicked ( objeto remitente, EventArgs args)
        {
            button.Rotation = 0;
            button.AnchorY = radio / button.Height;
            esperar button.RotateTo (360, 1000);
        }
    }
```

los Botón luego hace una rotación completa de 360 grados alrededor del centro de la página. Aquí está en curso:



funciones de aceleración

Ya ha visto la siguiente animación fotograma clave que se balancea la Botón un lado y luego al otro:

```
vacio asíncrono OnButtonClicked ( objeto remitente, EventArgs args)
{
    esperar button.RotateTo (90, 250);
    esperar button.RotateTo (-90, 500);
    esperar button.RotateTo (0, 250);
}
```

Pero la animación no acaba ve bien. El movimiento parece muy mecánico y robótico porque las rotaciones tienen una velocidad angular constante. En caso de que no la Botón al menos reducir la velocidad, ya que invierte la dirección y luego acelerar de nuevo?

Puede controlar los cambios de velocidad en animaciones con el uso de funciones de aceleración. Tras observar un par de funciones de aceleración de fabricación casera en el capítulo 21, "transforma." Xamarin.Forms incluye una clase que le permite especificar una función de transferencia simple que controla la velocidad animaciones o frenar, ya que está en funcionamiento.

Usted recordará que las animaciones generalmente implican una variable llamada *t* o *Progreso* que aumenta de 0 a 1 en el transcurso de la animación. Esta *t* variable se utiliza entonces en una interpolación entre *d* y *a* valores:

$$\text{.....} = \text{.....} + \cdot \cdot \cdot (\text{.....} - \text{.....})$$

La función de aceleración introduce una pequeña función de transferencia en este cálculo:

$$\text{.....} = \text{.....} + \text{.....} (\cdot) \cdot (\text{.....} - \text{.....})$$

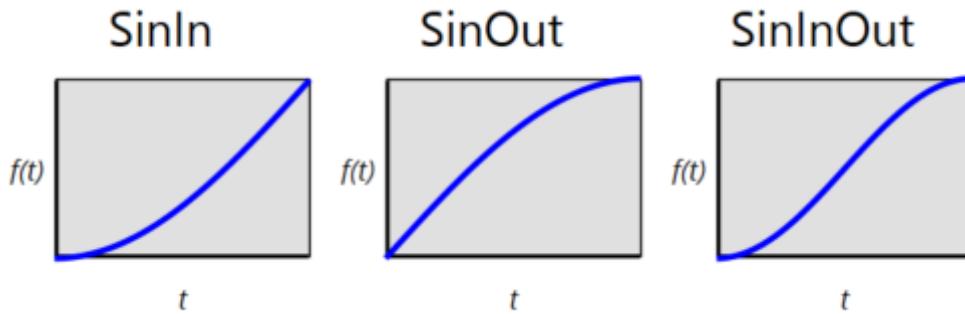
los aliviar clase define un método llamado Facilitar que lleva a cabo este trabajo. Para una entrada de 0, la Facilitar método devuelve 0, y para una entrada de 1, Facilitar devuelve 1. Entre esos dos valores, algo de matemáticas -a menudo una vez *minúsculo* trozo de matemáticas-da la animación una velocidad no constante. (Como se verá más adelante, no es del todo necesario que el Facilitar método asigne 0 hasta 0 y 1 a 1, pero eso es ciertamente el caso normal).

Puede definir sus propias funciones de aceleración, pero el aliviar clase define 11 campos de sólo lectura estática de tipo aliviar por su conveniencia:

- lineal (el valor por defecto)
- SinIn, SinOut, y SinInOut
- CubicIn, CubicOut, y CubicInOut
- BounceIn y Rebotar fuera
- Springin y Brotará

los En y Fuera sufijos indican si el efecto es prominente en el inicio de la animación, al final, o ambos.

los SinIn, SinOut, y SinInOut funciones de aceleración se basan en las funciones seno y coseno:



En cada uno de estos gráficos, el eje horizontal es el tiempo lineal, de izquierda a derecha desde 0 a 1. El eje vertical muestra la salida de la Facilitar método, 0 a 1 de abajo a arriba. Una pendiente más pronunciada vertical, es más rápido, mientras que una pendiente más horizontal es más lento.

los SinIn es el primer cuadrante de una curva coseno pero restado de 1 por lo que va de 0 a 1; que comienza lento, pero se hace más rápido. los SinOut es el primer cuadrante de una curva sinusoidal, partiendo algo más rápido que una animación lineal sino ralentizar hacia el final. los SinInOut es el primer medio de una curva coseno (de nuevo ajustado a ir de 0 a 1); es lento al principio y al final.

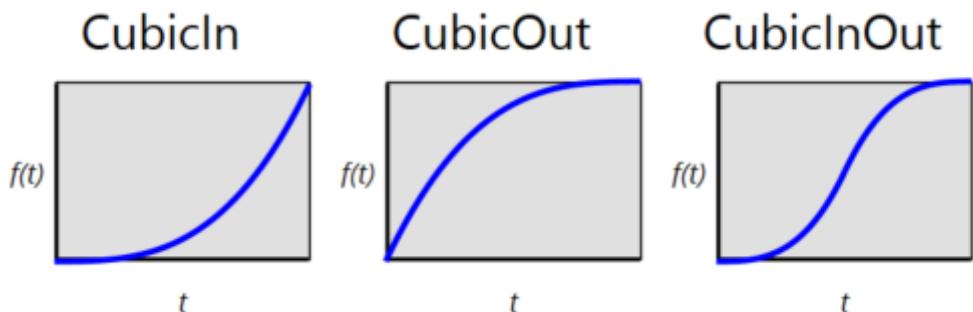
Debido a un movimiento armónico se describe mejor por las curvas sinusoidales, estas funciones de aceleración son ideales para una

Botón balanceándose hacia adelante y atrás. Se puede especificar un objeto de tipo aliviar como el último argumento de la RotateTo métodos:

```
vacío asíncrono OnButtonClicked ( objeto remitente, EventArgs args)
{
    esperar button.RotateTo (90, 250, aliviar .SinOut);
    esperar button.RotateTo (-90, 500, aliviar .SinInOut);
    esperar button.RotateTo (0, 250, aliviar .SinIn);
}
```

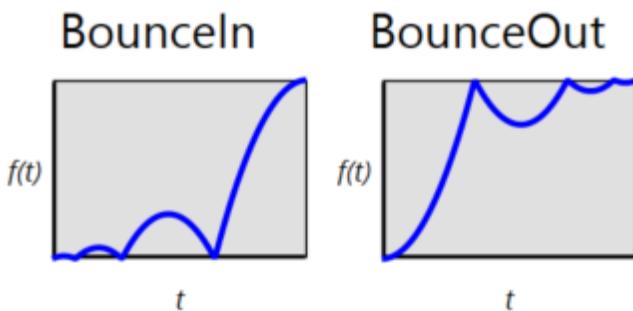
Y ahora el movimiento es mucho más natural. los Botón se ralentiza a medida que se acerca el momento en que se invierte el movimiento y luego se acelera de nuevo.

los CubicIn función de aceleración es simplemente la entrada elevado a la tercera potencia. los CubicOut es el reverso de eso, y CubicInOut combina los dos efectos:



La diferencia de velocidad es más acentuada que la flexibilización de seno.

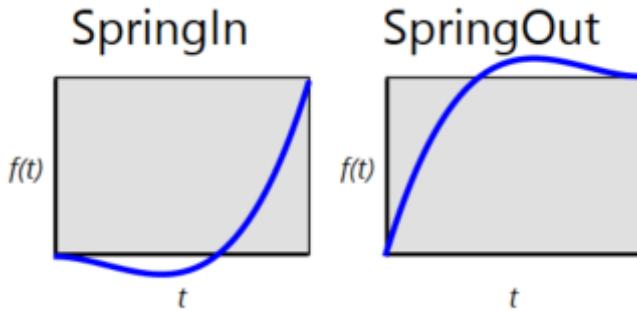
los BounceIn y Rebotar fuera rebotar al principio o al final, respectivamente:



Como se puede imaginar, la Rebotar fuera es ideal para la animación de las transformaciones que parecen tropezar con un obstáculo.

La salida de la Springin y Brotará funciones realidad van más allá del rango de 0 a 1. El Springin tiene una salida que cae por debajo de 0 inicialmente, y el Brotará salida va más allá de la

valor de 1:



En otros sistemas de animación, estos Springin y Brotarán patrones generalmente se conocen como *back-facilidad* funciones, y que visto la matemática subyacente en el **BoxViewClock** muestra en el capítulo anterior. De hecho, se puede reescribir la Convertir método en el **SecondBackEaseConverter** de esta manera y que funcionará de la misma:

```

público objeto Convertir( objeto valor, Tipo tipo de objetivo,
                           objeto parámetro, CultureInfo cultura)
{
    En t segundos = ( En t ) ( ( doble ) Valor / 6 );
    doble t = ( doble ) Valor / 6% 1;
    doble v = 0;

    Si ( T < 0,5 )
    {
        v = 0,5 * aliviar .SpringIn.Ease ( 2 * t );
    }
    más
    {
        v = 0,5 * (1 + aliviar .SpringOut.Ease ( 2 * ( t - 0,5 ) ));

    }

    regreso 6 * (segundo + v);
}

```

No hay **SpringInOut** objeto, por lo que la Convertir método debe romper cada segundo en dos mitades. Cuando t es inferior a 0,5, la Springin se aplica objeto. Sin embargo, la entrada a la Facilitar método necesita ser duplicado a variar de 0 a 1, y la salida tiene que ser reducido a la mitad a la gama de 0 a 0,5. los Brotarán llamada debe ajustarse asimismo: Cuando t varía de 0,5 a 1, la entrada a la Facilitar método necesita variar de 0 a 1, y la salida tiene que ser ajustada a intervalo de 0,5 a 1.

Vamos a probar algunas de las funciones más alivio. los **BounceButton** programa tiene un archivo XAML que es lo mismo que **AnimationTryout**, y el hecho clic controlador para el Botón tiene sólo tres estados:

```

público clase parcial BounceButtonPage : Pagina de contenido
{
    público BounceButtonPage ()
}

```

```
{  
    InitializeComponent();  
}  
  
vacío asíncrono OnButtonClicked ( objeto remitente, EventArgs args)  
{  
    esperar button.TranslateTo (0, (Altura - button.Height) / 2, 1000, aliviar .Rebotar fuera);  
    esperar Tarea .Delay (2000);  
    esperar button.TranslateTo (0, 0, 1000, aliviar .Brotará);  
}  
}
```

los Traducir a método anima el TranslationX y TranslationY propiedades. Los dos primeros argumentos se nombran X y Y, e indican los valores finales que se establecen en TranslationX y

TranslationY. El primero Traducir a llamar aquí no se mueve el Botón horizontalmente, por lo que el primer argumento es 0. El segundo argumento es la distancia entre la parte inferior de la Botón y la parte inferior de la página. los Botón está centrado verticalmente en la página, de modo que la distancia es la mitad de la altura de la página menos la mitad de la altura de la Botón.

Esa primera animación se realiza en 1.000 milisegundos. Luego hay un retardo de dos segundos, y el Botón se traduce de nuevo a su posición original con X y y argumentos de 0. El segundo Trans- tarde para animación utiliza el Easing.SpringOut función, por lo que probablemente la espera Botón a rebasar su marca y, a continuación asentarse de nuevo en su posición final.

sin embargo, el Traducir a Método enclava la salida de cualquier función de aceleración que va fuera del rango de 0 a 1. Más adelante en este capítulo verá una solución para que la falla en el Traducir a método.

Sus propias funciones de aceleración

Es fácil de hacer sus propias funciones de aceleración. Todo lo que se necesita es un método de tipo Func <doble, doble>, que es una función con una doble argumento y una doble valor de retorno. Se trata de una función de transferencia: Se debe devolver 0 para un argumento de 0 y 1 para un argumento de 1. Pero entre esos dos valores, todo vale.

Generalmente se le define una función de aceleración a medida que el argumento de la aliviar constructor. Ese es el único constructor aliviar define, pero el aliviar clase también define una conversión implícita de una Func <dobles, dobles> a Aliviar.

Las funciones de animación Xamarin.Forms llaman Facilitar método de la aliviar objeto. Ese Facilitar método también tiene una doble argumento y una doble valor de retorno, y que básicamente proporciona acceso público a la función de aceleración se especifica en el aliviar constructor. (Los gráficos anteriores en este capítulo que mostraron las diversas funciones de aceleración predefinidas fueron generados por un programa que accede a la Facilitar métodos de los diversos predefinido aliviar objetos.)

He aquí un programa que incorpora dos funciones de aceleración personalizada para controlar el escalado de una Botón. Estas funciones un tanto en contra del sentido de la palabra "facilidad", por lo que el programa es

llamado **UneasyScale**. La primera de estas dos funciones de aceleración trunca el valor de entrada a los valores discretos de 0, 0,2, 0,4, 0,6, 0,8, y 1, por lo que la Botón aumenta su tamaño en los saltos. Los Botón a continuación, se reduce en tamaño con otra función de aceleración que se aplica una pequeña variación al azar al valor de entrada.

La primera de estas funciones de aceleración se especifica como un argumento de función lambda para el aliviar constructor. El segundo es un método de fundido a una aliviar objeto:

```
público clase parcial UneasyScalePage : Pagina de contenido
{
    Aleatorio = aleatorios nuevo Aleatorio ();

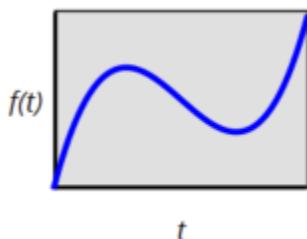
    público UneasyScalePage ()
    {
        InitializeComponent ();
    }

    vacío asíncrono OnButtonClicked ( objeto remitente, EventArgs args )
    {
        doble escala = Mates .min (Ancho / button.Width, Altura / button.Height);
        esperar button.ScaleTo (escala, 1000, nuevo aliviar (t => ( En t ) (5 * t) / 5,0));
        esperar button.ScaleTo (1, 1000, ( aliviar ) RandomEase);
    }

    doble RandomEase ( doble t )
    {
        regreso t == 0 || t == 1? t: t + 0.25 * (random.NextDouble () - 0.5);
    }
}
```

Por desgracia, es más fácil hacer las funciones inconexas como estos en lugar de funciones de transferencia más suave y más interesantes. Aquellos que tienden a ser necesariamente un poco más complejo.

Por ejemplo, supongamos que desea una función de aceleración que tiene este aspecto:



Comienza rápido, entonces se ralentiza y revierte supuesto, pero luego invierte el curso de nuevo a subir rápidamente en el tramo final.

Se puede suponer que se trata de una ecuación polinómica, o por lo menos que se puede aproximar por una ecuación polinómica. Tiene dos puntos en los que la pendiente es cero, lo que sugiere, además, que este es un cúbico

y puede representarse así:

$$\bullet(\cdot) = \bullet \bullet \bullet 3 + \bullet \bullet \bullet 2 + \bullet \bullet \bullet + \bullet$$

Ahora todo lo que tenemos que encontrar son los valores de *un*, *segundo*, *do*, y *re* que hará la función de transferencia se comporte como nosotros queremos.

Para los puntos finales, sabemos que:

$$\bullet(0) = 0 \bullet$$

$$(1) = 1$$

Esto significa que:

$$\bullet = 0$$

y:

$$1 = \bullet + \bullet + \bullet$$

Si decimos, además, que las dos caídas en la curva están en *igual* a $1/3$ y $2/3$, y los valores de *pie*) en esos puntos son $2/3$ y $1/3$, respectivamente, entonces:

$$\frac{2}{3} = \bullet \bullet \underline{1} 27 + \bullet \bullet \underline{1} 9 + \bullet \bullet \underline{1} 3$$

$$\frac{1}{3} = \bullet \bullet \underline{8} 27 + \bullet \bullet \underline{4} 9 + \bullet \bullet \underline{2} 3$$

Estas dos ecuaciones son algo más legible y manipulable si se convierten en coeficientes enteros, así que lo que tenemos son tres ecuaciones con tres incógnitas:

$$1 = \bullet + \bullet + \bullet$$

$$18 = \bullet + 3 \bullet + 9 \bullet$$

$$9 = 8 \bullet + 12 \bullet + 18 \bullet$$

Y con un poco de manipulación y combinación y el trabajo, se puede encontrar *un*, *segundo*, y *do*:

$$\bullet = 9$$

$$\bullet = -\frac{27}{2}$$

$$\bullet = \frac{11}{2}$$

Vamos a ver si hace lo que creemos que va a hacer. Los **CustomCubicEase** programa tiene un archivo XAML que es lo mismo que los proyectos anteriores. La función de aceleración se expresa aquí directamente como `Func<double, double>` objeto de modo que se puede utilizar convenientemente en dos `scaleTo` llamadas. Los Botón se escala por primera vez en tamaño, y luego después de una pausa de un segundo, el Botón está reducido a la normalidad:

```
pública clase parcial CustomCubicEasePage : Pagina de contenido
```

```

{
    público CustomCubicEasePage ()
    {
        InitializeComponent ();
    }

    vacio asíncrono OnButtonClicked (objeto remitente, EventArgs args)
    {
        Func < doble , doble > CustomEase = t => 9 * t * t * t - 13,5 * t * t + 5,5 * t;

        doble escala = Mates .min (Ancho / button.Width, Altura / button.Height);
        esperar button.ScaleTo (escala, 1000, CustomEase);
        esperar Tarea .Delay (1000);
        esperar button.ScaleTo (1, 1000, CustomEase);
    }
}

```

Si no tenemos en cuenta el trabajo de hacer sus propias funciones de aceleración para ser "divertido y relajante," una buena fuente para muchas funciones de aceleración estándar es el sitio web <http://robertpenner.com/easing/>.

También es posible construir funciones de aceleración de Math.sin y Math.cos si necesita un movimiento armónico simple y combinar los que tienen math.exp para incrementos exponenciales o caries.

Tomemos un ejemplo: Supongamos que desea una Botón que, cuando se hace clic, se abre hacia abajo desde la esquina inferior izquierda de su, casi como si el Botón eran una imagen fija a la pared con un par de clavos, y una de las uñas se cae, para que la imagen se desliza hacia abajo y se bloquee por un solo clavo en su esquina inferior izquierda.

Puede seguir junto con este ejercicio en el **AnimationTryout** programa. En el hecho clic controlador para el Botón, vamos a empezar poniendo el **anchorX** y **anchorY** propiedades y luego llamada **RotateTo**

para un giro de 90 grados:

```

button.AnchorX = 0;
button.AnchorY = 1;
esperar button.RotateTo (90, 3000);

```

Aquí está el resultado de que la animación cuando se ha completado:



Pero esto en realidad clama por una función de aceleración para que el Botón balancea hacia atrás y hacia adelante un poco de esa esquina antes de establecerse.

Para empezar, primero vamos a añadir una función de aceleración lineal que no hace nada a la Girar-

A llamada:

```
esperar button.RotateTo (90, 3000, nuevo aliviar (T => t));
```

Ahora vamos a añadir un comportamiento sinusoidal. Eso es, ya sea un seno o un coseno. Queremos que la oscilación sea lenta al principio, por lo que implicaría un coseno en lugar de un seno. Vamos a establecer el argumento de la

Math.cos método de modo que como t va de 0 a 1, el ángulo es de 0 a 10π . Eso es cinco ciclos completos de la curva coseno, lo que significa que el Botón balancea cinco veces de ida y vuelta:

```
esperar button.RotateTo (90, 3000, nuevo aliviar (T => Mates .Cos (10 * Mates t.PI *)));
```

Por supuesto, esto no es correcto en absoluto. Cuando t es cero, el **Math.cos** método devuelve 1, por lo que la animación comienza saltando a un valor de 90 grados. Para los valores subsiguientes de t , el **Math.cos** función devuelve valores que van de 1 a -1, por lo que la Botón balancea cinco veces a partir de 90 grados a -90 grados y de nuevo a 90 grados, llegando finalmente a un descanso de 90 grados. Esta es sin duda donde queremos que termine la animación, pero queremos que comience la animación a 0 grados

Sin embargo, vamos a ignorar el problema por un momento. En lugar de eso frente a lo que inicialmente parece ser el problema más complejo. No queremos que la Botón para girar 180 grados completos cinco veces. Queremos que los vaivenes del Botón a decaer con el tiempo antes de que llegue a descansar.

Hay una manera fácil de hacer eso. Podemos multiplicar el **Math.cos** método por una **math.exp** llamar con un argumento negativo sobre la base de t :

```
Mates .exp (-5 * t)
```

Los **math.exp** método plantea la constante matemática e (aproximadamente 2,7) a la potencia especificada.

Cuando t es 0 en el inicio de la animación, $\sin t$ a la potencia 0 es 1. Y cuando t es 1, $\sin t$ a la quinta potencia negativa es inferior a 0,01, que es muy cercana a cero. (No es necesario utilizar -5 en la presente convocatoria, se puede experimentar para encontrar un valor que parece mejor.)

Vamos a multiplicar el `Math.cos` como resultado de la `math.exp` resultado:

```
esperar button.RotateTo (90, 3000, nuevo aliviar (T => Mates .Cos (10 * Mates t .PI *) * Mates .exp (-5 * t)));
```

Estamos muy, muy cerca. los `math.exp` en efecto amortiguar el `Math.cos` llamar, pero el producto es al revés El producto es 1 cuando t es 0 y casi 0 cuando t 1. podemos solucionar esto simplemente restando toda la expresión a partir del 1? Vamos a intentarlo:

```
esperar button.RotateTo (90, 3000,  
nuevo aliviar (T => 1 - Mates .Cos (10 * Mates t .PI *) * Mates .exp (-5 * t)));
```

Ahora la función de aceleración vuelve correctamente 0 cuando t es 0, y lo suficientemente cerca a 1 cuando t es 1.

Y, lo que es más importante, la función de aceleración es ahora satisfactoria visualmente también. Realmente parece como si el Botón cae de su amarre y columpios varias veces antes de llegar al descanso.

Ahora vamos a llamar `Traducir a` para hacer que el Botón dejar y caen al fondo de la página. ¿Qué distancia Botón necesitará gata?

los Botón fue colocada originalmente en el centro de la página. Eso significa que la distancia entre la parte inferior de la Botón y la página era la mitad de la altura de la página menos la altura de la

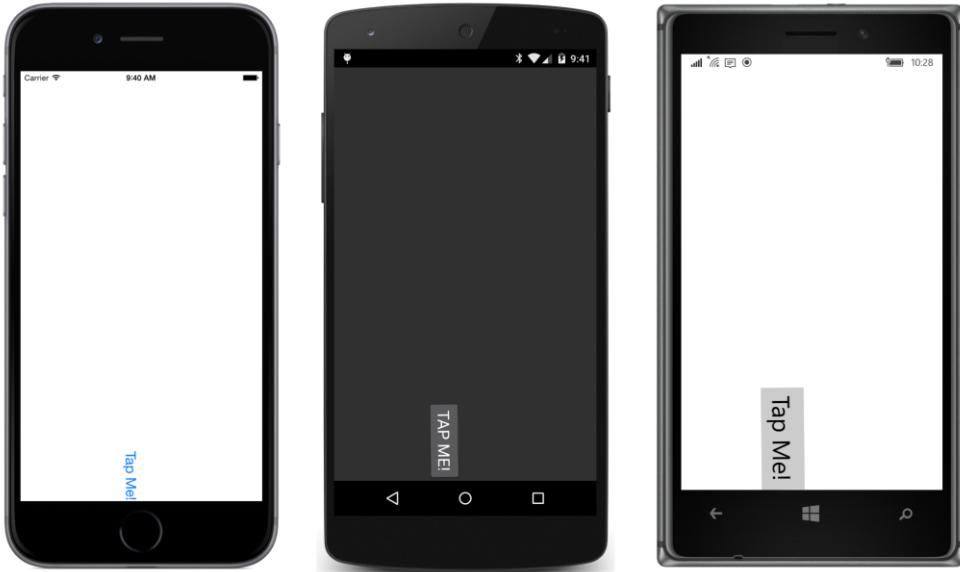
Botón:

`(Altura - button.Height) / 2`

Pero ahora el Botón ha girado 90 grados desde su esquina inferior izquierda, por lo que la Botón está más cerca de la parte inferior de la página por su anchura. Aquí está la llamada a plena `Traducir a` dejar caer la Botón a la parte inferior de la página y hacer rebotar un poco:

```
esperar button.TranslateTo (0, (Altura - button.Height) / 2 - button.Width,  
1000, aliviar .Rebotar fuera);
```

los Botón queda en reposo como esto:

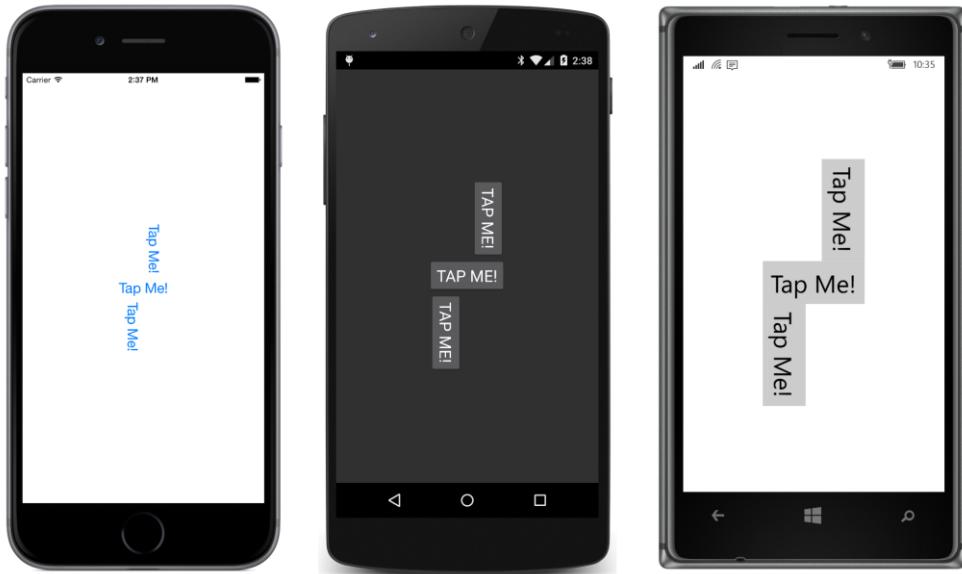


Ahora vamos a hacer el Botón quilla sobre la tierra y al revés, lo que significa que queremos hacer girar el Botón vuelta de la esquina superior derecha. Esto requiere un cambio en la anchorX y anchorY propiedades:

```
button.AnchorX = 1;  
button.AnchorY = 0;
```

Pero eso es un problema, una *grande* problemas debido a que un cambio en la anchorX y anchorY propiedades realmente cambia la ubicación de la Botón. ¡Intentalo! los Botón De repente salta hacia arriba y hacia la derecha. Donde el Botón salta a la posición es exactamente lo que sería si la primera RotateTo se había basado en estos nuevos anchorX y anchorY Los valores-una rotación alrededor de su esquina superior derecha en lugar de su esquina inferior izquierda.

Se puede visualizar que? He aquí una pequeña maqueta que muestra la posición original de la Botón, el Botón giran 90 grados en sentido horario desde su esquina inferior izquierda, y el Botón giran 90 grados en sentido horario desde su esquina superior derecha:



Cuando establecimos nuevos valores de `anchorX` y `anchorY`, tenemos que ajustar el `TranslationX` y `TranslationY` propiedades para que la Botón mueve esencialmente de la posición girada en la parte superior derecha a la posición girada en la inferior izquierda. `TranslationX` necesita ser disminuido por el ancho de la Botón y luego aumentado en su altura. `TranslationY` necesita ser incrementado tanto por la altura de la Botón y la anchura de la Botón. Vamos a tratar de que:

```
button.TranslationX -= button.Width - button.Height;
button.TranslationY += button.Width + button.Height;
```

Y que conserva la posición de la Botón cuando el `anchorX` y `anchorY` propiedades se cambian a esquina superior derecha del botón.

Ahora el Botón se puede girar alrededor de su esquina superior derecha a medida que cae encima, con otro poco de rebote, por supuesto:

[esperar button.RotateTo \(180, 1000, aliviar .Rebotar fuera\);](#)

Y ahora el Botón puede ascender hasta la pantalla y al mismo tiempo se desvanece:

```
esperar Tarea .Cuando todo
{
    button.FadeTo (0, 4000),
    button.TranslateTo (0, -Altura, 5000, aliviar .CubicIn)
};
```

Los Desvanecerse hacia método anima el Opacidad propiedad, en este caso de su valor predeterminado de 1 a 0 el valor especificado como el primer argumento.

Aquí está el programa completo, llamado **SwipeButton** (refiriéndose a la primera animación) y concluyendo con una restauración de la Botón a su posición original para que pueda intentarlo de nuevo:

```

pública clase parcial SwingButtonPage : Pagina de contenido
{
    pública SwingButtonPage ()
    {
        InitializeComponent ();
    }

    vacío asíncrono OnButtonClicked ( objeto remitente, EventArgs args)
    {
        // El columpio hacia abajo desde la esquina inferior izquierda.
        button.AnchorX = 0;
        button.AnchorY = 1;

        esperar button.RotateTo (90, 3000,
            nuevo aliviar (T => 1 - Mates .Cos (10 * Mates t .Pi *) * Mates .exp (-5 * t));

        // caen al fondo de la pantalla.
        esperar button.TranslateTo (0, (Altura - button.Height) / 2 - button.Width,
            1000, aliviar .Rebotar fuera);

        // Preparar anchorX y anchorY para la próxima rotación.
        button.AnchorX = 1;
        button.AnchorY = 0;

        // compensar el cambio en anchorX y anchorY.
        button.TranslationX -= button.Width - button.Height;
        button.TranslationY += button.Width + button.Height;

        // Caerse.
        esperar button.RotateTo (180, 1000, aliviar .Rebotar fuera);

        // Fade out mientras asciende a la parte superior de la pantalla.
        esperar Tarea .Cuando todo
        (
            button.FadeTo (0, 4000),
            button.TranslateTo (0, -Altura, 5000, aliviar .CubicIn)
        );

        // Despues de tres segundos, devolver el botón a la normalidad.
        esperar Tarea .Delay (3000);
        button.TranslationX = 0;
        button.TranslationY = 0;
        button.Rotation = 0;
        button.Opacity = 1;
    }
}

```

Una función de aceleración se supone que debe devolver 0 cuando la entrada es 0 y 1 cuando la entrada es 1, pero es posible romper estas reglas, ya veces eso tiene sentido. Por ejemplo, supongamos que desea una animación que se mueve un elemento un poco, tal vez Vibra de alguna manera, pero la animación debe devolver el elemento a su posición original al final. Para algo como esto tiene sentido para la función de aceleración para volver 0 cuando la entrada es al mismo tiempo 0 y 1, pero algo distinto de 0 entre esos valores.

Esta es la idea detrás JiggleButton, que está en el **Xamarin.FormsBook.Toolkit** biblioteca. Plantilla-JiggleButton deriva de Botón e instala una hecho clic controlador para el único propósito de balanceándose en el botón al hacer clic en ella:

```
espacio de nombres Xamarin.FormsBook.Toolkit
{
    clase pública JiggleButton : Botón
    {
        bool isJiggling;

        público JiggleButton ()
        {
            Seguido += asincrono (Remitente, args) =>
            {
                Si (IsJiggling)
                    regreso ;

                isJiggling = cierto ;

                esperar a este .RotateTo (15, 1000, nuevo aliviar (T =>
                    Mates .Pecado( Mates t .PI *) *
                    Mates .Pecado( Mates .PI * 20 * t));
                isJiggling = falso ;
            };
        }
    }
}
```

los RotateTo método parece girar el botón en 15 grados a lo largo de un segundo. Sin embargo, la costumbre aliviar objeto tiene una idea diferente. Consiste únicamente en el producto de dos funciones seno. Como t va de 0 a 1, la primera Math.sin función barre la primera mitad de una curva sinusoidal, por lo que va de 0 cuando t es 0, a 1 cuando t es 0,5, y de nuevo a 0 cuando t es 1.

El segundo Math.sin llamada es la parte agitan. Como t va de 0 a 1, esta llamada pasa a través de 10 ciclos de una curva sinusoidal. Sin la primera Math.sin llamar, esto sería girar el botón de 0 a 15 grados, y luego a -15 grados, y de nuevo a 0 diez veces. Pero la primera Math.sin llamada amortigua que la rotación al principio y al final de la animación, permitiendo sólo un 15 y -15 grados rotación completa en el medio.

Un poco de código que implica la isJiggling protege el campo hecho clic manejador de comenzar una nueva animación cuando uno ya está en curso. Esto es una ventaja de usar esperar con los métodos de animación: Usted sabe exactamente cuando se ha completado la animación.

los **JiggleButtonDemo** archivo XAML crea tres JiggleButton objetos para que pueda jugar con ellos:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    xmlns: kit de herramientas =
        "clr-espacio de nombres: Xamarin.FormsBook.Toolkit; montaje = Xamarin.FormsBook.Toolkit "
    x: Clase = " JiggleButtonDemo.JiggleButtonDemoPage " >
```

```

< StackLayout >
    < kit de herramientas: JiggleButton Texto = "Puntee en Yo!" 
        Tamaño de fuente = "Grande"
        HorizontalOptions = "Centrar"
        VerticalOptions = "CenterAndExpand" />

    < kit de herramientas: JiggleButton Texto = "Puntee en Yo!" 
        Tamaño de fuente = "Grande"
        HorizontalOptions = "Centrar"
        VerticalOptions = "CenterAndExpand" />

    < kit de herramientas: JiggleButton Texto = "Puntee en Yo!" 
        Tamaño de fuente = "Grande"
        HorizontalOptions = "Centrar"
        VerticalOptions = "CenterAndExpand" />

</ StackLayout >
</ Página de contenido >

```

animaciones de entrada

Un tipo común de la animación en la programación de la vida real se produce cuando una página se hace visible por primera vez. Los diversos elementos de la página se pueden animar brevemente antes de establecerse en sus estados finales. Esto se llama a menudo una *animación de entrada* y puede implicar:

- La traducción, para mover elementos en sus posiciones finales.
- Escala, para ampliar o reducir elementos a sus tamaños finales.
- Cambios en Opacidad a desvanecerse elementos a la vista.
- la rotación 3D para hacer que parezca como si una página entera se balancea a la vista.

En general, usted querrá los elementos de la página a quedar en reposo con los valores predeterminados de estas propiedades: TranslationX y TranslationY valores de 0, Escala y Opacidad valores de 1, y todos Lista-
ción propiedades establecen en 0.

En otras palabras, las animaciones de entrada debe *fin* a su valor predeterminado de cada propiedad, lo que significa que comienzan a valores no predeterminados. Este enfoque también permite que el programa para aplicar otras transformadas a estos elementos en un momento posterior sin tomar las animaciones de ingreso en cuenta.

Al diseñar la disposición en XAML que querrá simplemente ignorar estas animaciones. A modo de ejemplo, he aquí una página con varios elementos exclusivamente para fines de demostración. El programa se llama FadingEntrance:

```

< Página de contenido xmlns = "http://xamarin.com/schemas/2014/forms" 
    xmlns: x = "http://schemas.microsoft.com/winfx/2009/xaml" 
    x: Class = "FadingEntrance.FadingEntrancePage" >

< ContentPage.Padding >
    < OnPlatform x: TypeArguments = "Espesor" 
        iOS = "10, 20, 10, 10" 
        Androide = "10" 
        WinPhone = "10" />

```

```
</ ContentPage.Padding >

< StackLayout x: Nombre = " stackLayout " >
    < Etiqueta Texto = " La aplicación "
        Estilo = " 0 DynamicResource TextStyle "
        FontAttributes = " Itálico "
        HorizontalOptions = " Centrar " />

    < Botón Texto = " cuenta regresiva "
        Tamaño de fuente = " Grande "
        HorizontalOptions = " Centrar " />

    < Etiqueta Texto = " Deslizador primaria "
        HorizontalOptions = " Centrar " />

    < deslizador Valor = " 0.5 " />

    < Vista de la lista HorizontalOptions = " Centrar "
        WidthRequest = " 200 " >
        < ListView.ItemsSource >
            < x: Array Tipo = " X: Tipo Color " >
                < Color > rojo </ Color >
                < Color > Verde </ Color >
                < Color > Azul </ Color >
                < Color > Agua </ Color >
                < Color > Púrpura </ Color >
                < Color > Amarillo </ Color >
            </ x: Array >
        </ ListView.ItemsSource >

        < ListView.ItemTemplate >
            < DataTemplate >
                < ViewCell >
                    < BoxView Color = " {Unión} " />
                </ ViewCell >
            </ DataTemplate >
        </ ListView.ItemTemplate >
    </ Vista de la lista >

    < Etiqueta Texto = " corredera secundaria "
        HorizontalOptions = " Centrar " />

    < deslizador Valor = " 0.5 " />

    < Botón Texto = " Lanzamiento "
        Tamaño de fuente = " Grande "
        HorizontalOptions = " Centrar " />
</ StackLayout >
</ Página de contenido >
```

El archivo de código subyacente anula la OnAppearing método. Los OnAppearing método se llama después de la página se presenta, pero antes de que la página se vuelve visible. Todos los elementos de la página han sido dimensionados y colocados, por lo que si usted necesita para obtener esa información puede hacerlo durante este método. En el

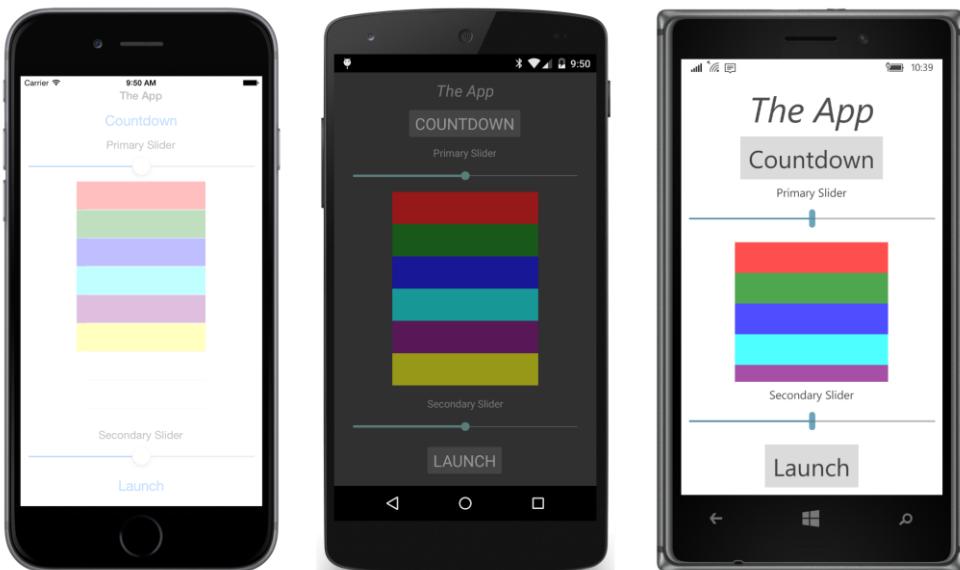
FadingEntrance programa, el **OnAppearing** override establece el **Opacidad** propiedad de la **StackLayout** a 0 (lo que hace todo dentro de la **StackLayout** invisible) y luego lo anima a 1:

```
pública clase parcial FadingEntrancePage : Pagina de contenido
{
    pública FadingEntrancePage ()
    {
        InitializeComponent ();
    }

    protegido override void OnAppearing ()
    {
        base .OnAppearing ();

        stackLayout.Opacity = 0;
        stackLayout.FadeTo (1, 3000);
    }
}
```

Aquí está la página en el proceso de decoloración a la vista:



Vamos a intentarlo otra. El archivo XAML en el **SlidingEntrance** programa es el mismo que **FadingEntrance**, pero el **OnAppearing** anulación comienza por toda la **TranslationX** propiedades de los hijos de la **StackLayout** a valores de 1,000 y -1,000 alterna:

```
pública clase parcial SlidingEntrancePage : Pagina de contenido
{
    pública SlidingEntrancePage ()
    {
        InitializeComponent ();
    }
```

```
asíncrono protegido override void OnAppearing ()  
{  
    base.OnAppearing ();  
  
    doble offset = 1,000;  
  
    para cada ( Ver ver en stackLayout.Children )  
    {  
        view.TranslationX = offset;  
        compensar * = -1;  
    }  
  
    para cada ( Ver ver en stackLayout.Children )  
    {  
        esperar Tarea .WhenAny (view.TranslateTo (0, 0, 1000, aliviar .Brotará),  
                               Tarea .Delay (100));  
    }  
}
```

El segundo para cada bucle y luego anima a estos niños volver a la configuración predeterminada de TranslationX y TranslationY. Sin embargo, las animaciones están escalonadas y se superponen. He aquí cómo: La primera llamada a Task.WhenAny comienza la primera Traducir a animación, que completa después de un segundo. Sin embargo, el segundo argumento Task.WhenAny es Task.Delay, que completa en una décima de segundo, y fue entonces cuando Task.WhenAny también completa. Los para cada bucle obtiene el siguiente hijo, que entonces comienza su propio uno-segundos de animación. Cada animación comienza una décima de segundo después de la anterior.

Aquí está el resultado en el proceso:



los Traducir a llamada utiliza la Easing.SpringOut función, lo que significa que cada elemento animado debe rebasar su destino y luego moverse hacia atrás para entrar en reposo en el centro de la página. Sin embargo, no verá que esto suceda. Como ya has descubierto, el Traducir a método deja de funcionar cuando una función de aceleración tiene una salida que excede 1.

Verás una solución para esto, y una versión de este programa con elementos que hayan rebasado destinos, más adelante en este capítulo.

Por último, he aquí una SwingingEntrance animación:

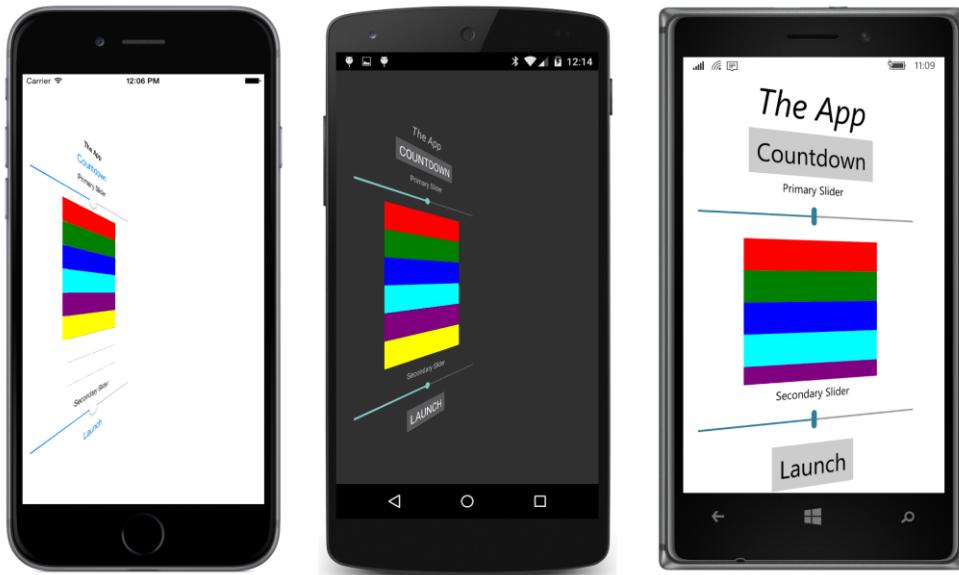
```
público clase parcial SwingingEntrancePage : Pagina de contenido
{
    público SwingingEntrancePage ()
    {
        InitializeComponent ();
    }

    asíncrono protected void override OnAppearing ()
    {
        base .OnAppearing ();

        stackLayout.AnchorX = 0;
        stackLayout.RotationY = 180;
        esperar stackLayout.RotateYTo (0, 1.000, aliviar .CubicOut);
        stackLayout.AnchorX = 0,5;
    }
}
```

los RotateYTo método gira el todo StackLayout y sus niños de todo el eje Y desde 180 grados a 0 grados. Con un anchorX ajuste de 0, la rotación es en realidad alrededor del borde izquierdo

del StackLayout. los StackLayout no serán visibles hasta que la rotationY valor es inferior a 90 grados, pero el resultado se ve un poco mejor si el giro se añade antes de la página en realidad se hace visible. los CubicOut función de aceleración hace que la animación a disminuir a medida que se acerca a la terminación. Aquí está en curso:



Después de la animación se ha completado, el OnAppearing devuelve el método anchorX a su valor original, de modo que todo tiene valores por defecto para las animaciones futuras que el programa puede ser que desee implementar.

animaciones para siempre

En el extremo opuesto de animaciones de entrada son *siempre animaciones*. Una aplicación puede aplicar una animación que sigue “siempre”, o al menos hasta que termine el programa. A menudo, el único propósito de tales animaciones es demostrar las capacidades de un sistema de animación, pero preferiblemente de una manera agradable o divertido.

El primer ejemplo se llama **FadingTextAnimation** y usos Desvanecerse hacia a desvanecerse dos Etiqueta elementos de entrada y salida. El archivo XAML pone a ambos Etiqueta elementos en una sola célula Cuadrícula para que se solapen. El segundo tiene su Opacidad propiedad pone a 0:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns:x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " FadingTextAnimation.FadingTextAnimationPage "
    Color de fondo = " Blanco "
    SizeChanged = " OnPageSizeChanged " >

< ContentPage.Resources >
    < ResourceDictionary >
        < Estilo Tipo de objetivo = " Etiqueta " >
```

```

<Setter Propiedad = "HorizontalTextAlignment" Valor = "Centrar" />
<Setter Propiedad = "VerticalTextAlignment" Valor = "Centrar" />
</Estilo>
</ResourceDictionary>
</ContentPage.Resources>

<Cuadrícula>
    <Etiqueta x: Nombre = "label1" 
        Texto = "MÁS"
        Color de texto = "Azul" />

    <Etiqueta x: Nombre = "label2" 
        Texto = "CÓDIGO"
        Color de texto = "rojo"
        Opacidad = "0" />
</Cuadrícula>
</Pagina de contenido>

```

Una manera simple de crear una animación que dirige “siempre” es poner todo el código que utilizan la animación esperar por supuesto-dentro de una mientras bucle con una condición de cierto. A continuación, llamar a ese método desde el constructor:

```

pública clase parcial FadingTextAnimationPage : Pagina de contenido
{
    pública FadingTextAnimationPage ()
    {
        InitializeComponent ();

        // Inicia la animación va.
        AnimationLoop ();
    }

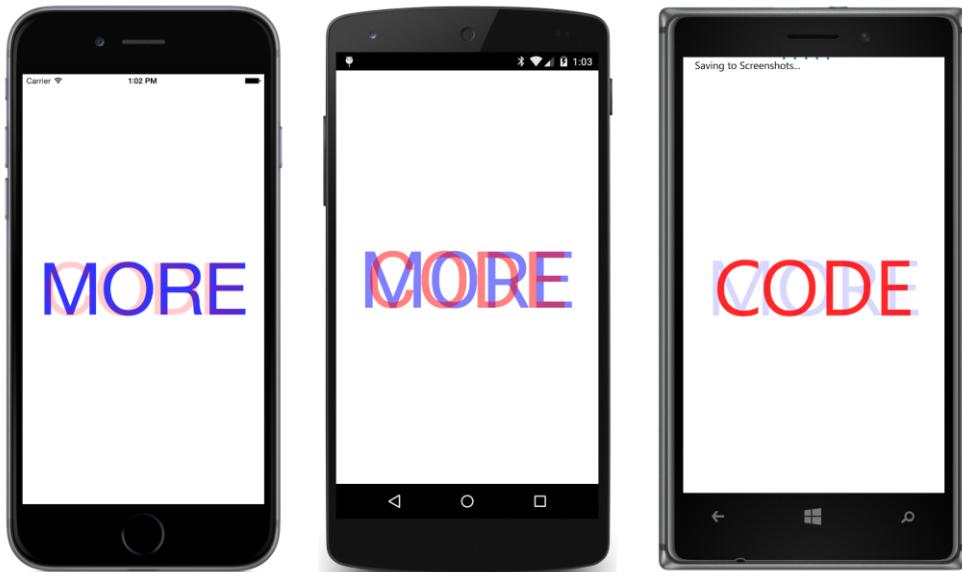
    vacío OnPageSizeChanged ( objeto remitente, EventArgs args)
    {
        Si (Ancho> 0)
        {
            doble fontSize = 0,3 * Anchura;
            label1.FontSize = fontSize;
            label2.FontSize = fontSize;
        }
    }

    vacío asincrono AnimationLoop ()
    {
        mientras ( cierto )
        {
            esperar Tarea .WhenAll (label1.FadeTo (0, 1000),
                label2.FadeTo (1, 1000));

            esperar Tarea .WhenAll (label1.FadeTo (1, 1000),
                label2.FadeTo (0, 1000));
        }
    }
}

```

bucles infinitos suelen ser peligrosos, pero éste ejecuta muy brevemente una vez por segundo cuando el Task.WhenAll método señala una terminación de las dos animaciones-la primera desvanecimiento uno Etiqueta y la segunda la decoloración en el otro Etiqueta. los SizeChanged manejador de la página establece el Tamaño de fuente del texto, por lo que el texto se acerca al ancho de la página:



Qué significa "Más código" o "Código más"? Tal vez ambas cosas.

Aquí hay otro que se dirige a la animación de texto. los **PalindromeAnimation** programa hace girar caracteres individuales 180 Degrees a su vez al revés. Afortunadamente, los personajes comprenden un palíndromo que se lee igual hacia adelante y hacia atrás:



Cuando todos los caracteres se da la vuelta al revés, todo el conjunto de caracteres se da la vuelta, y la animación se inicia de nuevo.

El archivo XAML simplemente contiene una horizontal StackLayout, sin hijos por el momento:

```
< Página de contenido xmlns = " http://kamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " PalindromeAnimation.PalindromeAnimationPage "
    SizeChanged = " OnPageSizeChanged " >

    < StackLayout x: Nombre = " stack_layout "
        Orientación = " Horizontal "
        HorizontalOptions = " Centrar "
        VerticalOptions = " Centrar "
        Espaciado = " 0 " />

</ Página de contenido >
```

El constructor del archivo de código subyacente llena este StackLayout con 17 Etiqueta elementos para deletrear la frase capicúa “NUNCA par o impar.” Al igual que en el programa anterior, el SizeChanged manejador de la página, ajusta el tamaño de estas etiquetas. Cada Etiqueta se le da un uniforme WidthRequest y una

Tamaño de fuente basado en que la anchura. Cada carácter en la cadena de texto debe ocupar el mismo ancho de modo que todavía están espaciados la misma cuando se dan la vuelta al revés:

```
pública clase parcial PalindromeAnimationPage : Página de contenido
{
    cuerda text = "NUNCA par o impar";
    doble [] AnchorX = {0,5, 0,5, 0,5, 1, 0,
        0,5, 1, -1,
        0,5, 1, 0,
        0,5, 0,5, 0,5};
```

```
públicoPalindromeAnimationPage ()  
{  
    InitializeComponent ();  
  
    // agregar una etiqueta a la StackLayout para cada carácter.  
    para ( En t i = 0; i <text.Length; i ++)  
    {  
        Etiqueta etiqueta = nuevo Etiqueta  
        {  
            Text = texto [i] .ToString (),  
            HorizontalTextAlignment = Alineación del texto .Centrar  
        };  
        stackLayout.Children.Add (etiqueta);  
    }  
  
    // Inicia la animación.  
    AnimationLoop ();  
}  
  
vacío OnPageSizeChanged ( objeto remitente, EventArgs args)  
{  
    // Ajuste el tamaño y la fuente basada en el ancho de la pantalla.  
    doble width = 0,8 * esta .Width / stackLayout.Children.Count;  
  
    para cada ( Etiqueta etiqueta en stackLayout.Children.OfType < Etiqueta > ())  
    {  
        label.FontSize = 1,4 * anchura;  
        label.WidthRequest = ancho;  
    }  
}  
  
vacío asíncrono AnimationLoop()  
{  
    bool hacia atrás = falso ;  
  
    mientras ( cierto )  
    {  
        // Vamos a quedarnos aquí un segundo.  
        esperar Tarea .Delay (1000);  
  
        // Preparar para las rotaciones superpuestas.  
        Etiqueta previousLabel = nulo ;  
  
        // Recorrer todas las etiquetas.  
        IEnumerable < Etiqueta > Etiquetas = stackLayout.Children.OfType < Etiqueta > ();  
  
        para cada ( Etiqueta etiqueta en al revés? labels.Reverse () : etiquetas)  
        {  
            uint flipTime = 250;  
  
            // Establecer el anchorX y propiedades anchorY.  
            En t index = stackLayout.Children.IndexOf (etiqueta);  
            label.AnchorX = anchorX [índice];  
            label.AnchorY = 1;
```

```

Si (PreviousLabel == nulo)
{
    // Para la primera etiqueta de la secuencia, girarlo 90 grados.
    esperar label.RelRotateTo (90, flipTime / 2);
}

más
{
    // Para la segunda y posterior, también terminar la tapa anterior.
    esperar Tarea.WhenAll (label.RelRotateTo (90, flipTime / 2),
                           previousLabel.RelRotateTo (90, flipTime / 2));
}

// Si es el último, terminar la tapa.
Si (Etiqueta == (hacia atrás labels.First () ?: Labels.Last ()))
{
    esperar label.RelRotateTo (90, flipTime / 2);
}

previousLabel = etiqueta;
}

// Girar toda la pila.
stackLayout.AnchorY = 1;
esperar stackLayout.RelRotateTo (180, 1000);

// Voltear la bandera al revés.
hacia atrás ^ = cierto;
}

}
}

```

Gran parte de la complejidad de la AnimationLoop resultados del método de animaciones superpuestas. Cada letra tiene que girar 180 grados. Sin embargo, los últimos 90 grados de cada rotación carta se superpone con los primeros 90 grados de la siguiente letra. Esto requiere que la primera letra y la última letra se manejan de manera diferente.

Las rotaciones de letras se complican aún más por la configuración de la anchorX y anchorY propiedades. Para cada rotación, anchorY se establece en 1 y la rotación se produce alrededor de la parte inferior de la Etiqueta.

Pero el entorno de la anchorX propiedad depende del lugar donde se produce la letra de la frase. Las cuatro primeras letras de "nunca" puede girar alrededor de la parte inferior central de la carta debido a que forman la palabra "incluso" cuando se invierte. Pero la "R" tiene que girar en torno a su esquina inferior derecha de modo que se convierte en el final de la palabra "O". El espacio después de "NUNCA" tiene que girar en torno a su esquina inferior izquierda de modo que se convierte en el espacio entre "O" y "par". En esencia, la "R" de "NUNCA" y los lugares de intercambio espacio. El resto de la frase continúa de manera similar. Los diversos anchorX valores para cada letra se almacenan en el

anchorX array en la parte superior de la clase.

Cuando todas las cartas han sido rotados de forma individual, el conjunto StackLayout se hace girar en 180 grados. A pesar de que hace girar StackLayout tiene el mismo aspecto como el StackLayout cuando el programa

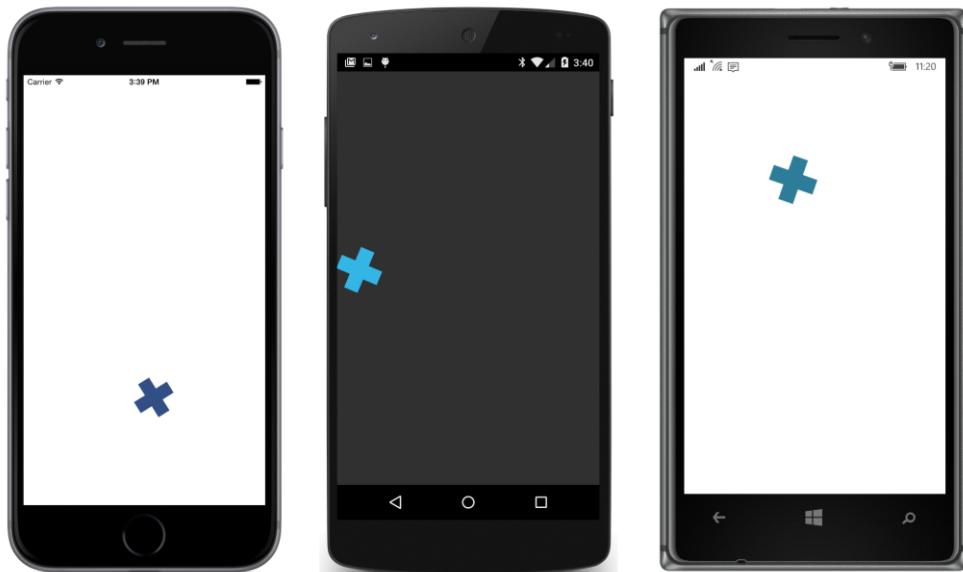
empezó a correr, no es lo mismo. La última letra de la frase es ahora el primer niño en el StackLayout fuera y la primera letra es ahora el último niño en el StackLayout. Esa es la razón de la hacia atrás variable. Los para cada instrucción utiliza para enumerar que a través de la StackLayout niños en una dirección hacia adelante o hacia atrás.

Se dará cuenta de que todo el anchorX y anchorY propiedades se establecen en el AnimationLoop justo antes de que se inicie la animación, a pesar de que nunca cambian en el transcurso del programa. Esto es para acomodar el problema con iOS. Las propiedades se deben establecer después de que el elemento ha sido de tamaño, y el establecimiento de las propiedades dentro de este bucle es simplemente conveniente.

Si no existiera ese problema con iOS, toda la anchorX y anchorY propiedades se podrían establecer en el constructor del programa o incluso en el archivo XAML. No es irrazonable para definir los 17 Etiqueta elementos en el archivo XAML con el único anchorX configuración de cada Etiqueta y lo común anchorY el establecimiento de una Estilo.

Como es, en dispositivos iOS, la **PalindromeAnimation** programa no puede sobrevivir a un cambio en la orientación de vertical a horizontal y viceversa. Después de la Etiqueta elementos cambian de tamaño, no hay nada que la aplicación puede hacer para solucionar el uso interno de la anchorX y anchorY propiedades.

los **CopterAnimation** programa simula un poco de vuelo del helicóptero en un círculo alrededor de la página. La simulación, sin embargo, es muy simple: El helicóptero es simplemente dos BoxView elementos dimensionados y dispuestos para parecerse alas:



El programa tiene dos rotaciones continuas. El que hace girar rápidamente palas del helicóptero alrededor de su centro. Una rotación más lenta mueve el conjunto de ala en un círculo alrededor del centro de la página. Ambas rotaciones utilizan el valor por defecto anchorX y anchorY configuración de 0,5, por lo que no hay problema en iOS.

Sin embargo, el programa utiliza implícitamente el ancho del teléfono de la circunferencia del círculo que las alas del helicóptero vuelan alrededor. Si se gira el teléfono hacia los lados a modo de paisaje, el helicóptero realmente volar fuera de los límites del teléfono.

El secreto de la simplicidad de **CopterAnimation** es el archivo XAML:

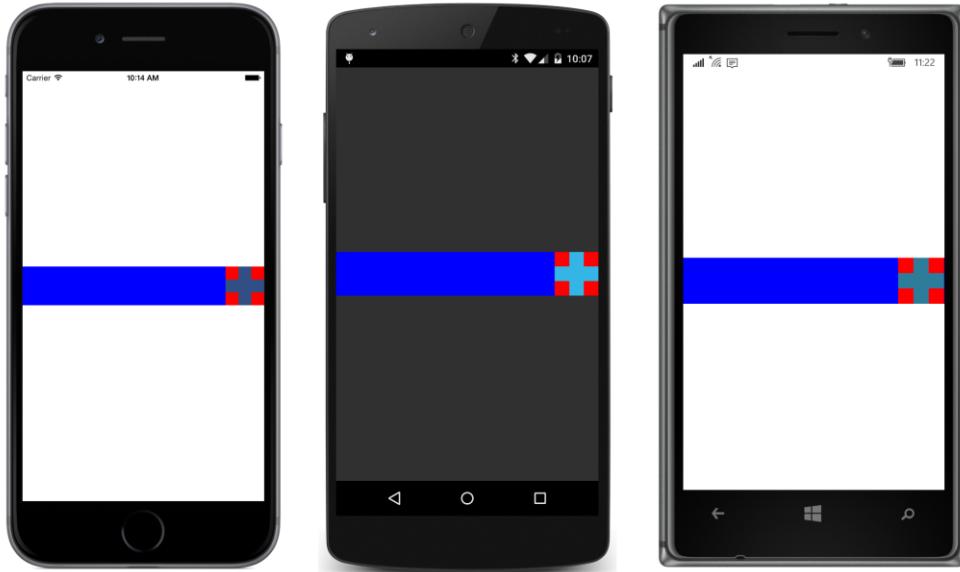
```
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " CopterAnimation.CopterAnimationPage " >

< ContentView x: Nombre = " revolveTarget "
    HorizontalOptions = " Llenar "
    VerticalOptions = " Centrar " >
    < ContentView x: Nombre = " copterView "
        HorizontalOptions = " Fin " >
        < AbsoluteLayout >
            < BoxView AbsoluteLayout.LayoutBounds = " 20, 0, 20, 60 "
                Color = " Acento " />

            < BoxView AbsoluteLayout.LayoutBounds = " 0, 20, 60, 20 "
                Color = " Acento " />
        </ AbsoluteLayout >
    </ ContentView >
</ ContentView >
</ Pagina de contenido >
```

El diseño completo consta de dos anidada **ContentView** elementos, con una **AbsoluteLayout** en el interior **ContentView** para los dos **BoxView** alas. El exterior **ContentView** (llamado **revolveTarget**) se extiende a la anchura del teléfono y se centra verticalmente en la página, pero sólo es tan alto como el interior **ContentView**. El interior **ContentView** (llamado **copterView**) está posicionado en el extremo derecho de la externa **ContentView**.

Probablemente se puede visualizar esto más fácilmente si se apaga la animación y dar los dos Contenido-Ver elementos de diferentes colores de fondo, por ejemplo, azul y rojo:



Ahora se puede ver con bastante facilidad que tanto éstos ContentView elementos se pueden girar alrededor de sus centros para conseguir el efecto de las alas que vuelan en un círculo de rotación:

```
public class parcial CopterAnimationPage : Pagina de contenido
{
    public CopterAnimationPage ()
    {
        InitializeComponent ();

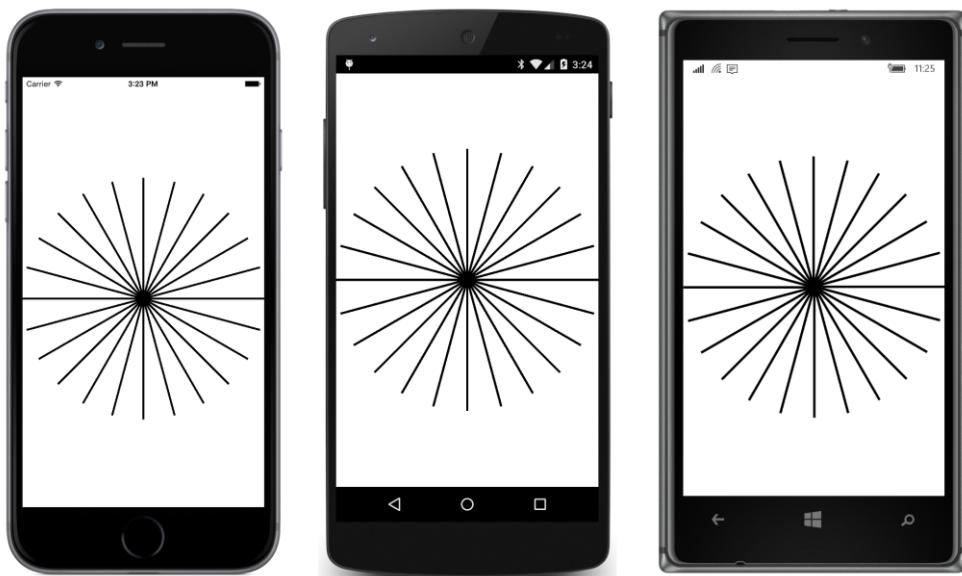
        AnimationLoop ();
    }

    vacío asíncrono AnimationLoop ()
    {
        mientras ( cierto )
        {
            revolveTarget.Rotation = 0;
            copterView.Rotation = 0;

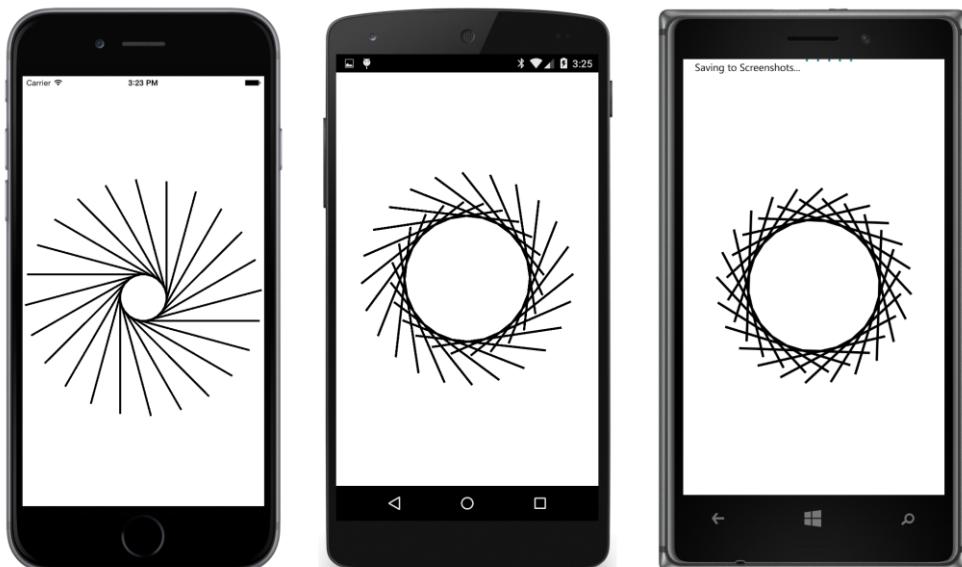
            esperar Tarea .WhenAll (revolveTarget.RotateTo (360, 5000),
                                      copterView.RotateTo (360 * 5, 5000));
        }
    }
}
```

Ambas animaciones tienen una duración de cinco segundos, pero durante ese tiempo, el exterior ContentView sólo gira una vez alrededor de su centro, mientras que el ensamblaje de las alas helicóptero gira cinco veces alrededor de su centro.

los **RotatingSpokes** programa dibuja 24 radios que emanan desde el centro de la página con una longitud basado en el menor de la altura y anchura de la página. Por supuesto, cada uno de los radios es una delgada BoxView elemento:



Después de tres segundos, el conjunto de radios comienza a girar alrededor del centro. Que se prolonga durante un rato, y luego cada radio individual empieza a girar en torno *sus* centro, que hace un patrón cambiante interesante:



Al igual que con **CopterAnimation**, el **RotatingSpokes** programa utiliza los valores predeterminados de `anchorX` y `UnChory` para todas las rotaciones, así que no hay problema cambiar la orientación de teléfono en los dispositivos IOS.

Pero el archivo XAML **RotatingSpokes** consiste únicamente en una `AbsoluteLayout` y sugiere nada

acerca de cómo funciona el programa:

```
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns:x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " RotatingSpokes.RotatingSpokesPage "
    Color de fondo = " Blanco "
    SizeChanged = " OnPageSizeChanged " >
< AbsoluteLayout x: Nombre = " AbsoluteLayout "
    HorizontalOptions = " Centrar "
    VerticalOptions = " Centrar " />
</ Pagina de contenido >
```

Todo lo demás está hecho en código. El constructor añade 24 negro BoxView elementos a la absoluteLayout, y el SizeChanged manejador de la página de los posiciona en el patrón Intervienen

```
pública clase parcial RotatingSpokesPage : Pagina de contenido
```

```
{
    const int numSpokes = 24;
    BoxView [] boxViews = nuevo BoxView [NumSpokes];

    pública RotatingSpokesPage ()
    {
        InitializeComponent ();

        // Cree todos los elementos BoxView.
        para ( En t i = 0; i <numSpokes; i++)
        {
            BoxView boxView = nuevo BoxView
            {
                color = Color .Negro
            };
            boxViews [i] = boxView;
            absoluteLayout.Children.Add (boxView);
        }

        AnimationLoop ();
    }

    vacío OnPageSizeChanged ( objeto remitente, EventArgs args)
    {
        // Conjunto AbsoluteLayout a una dimensión cuadrado.
        doble dimensión = Mates .min ( esta .Anchura, esta .Altura);
        absoluteLayout.WidthRequest = dimensión;
        absoluteLayout.HeightRequest = dimensión;

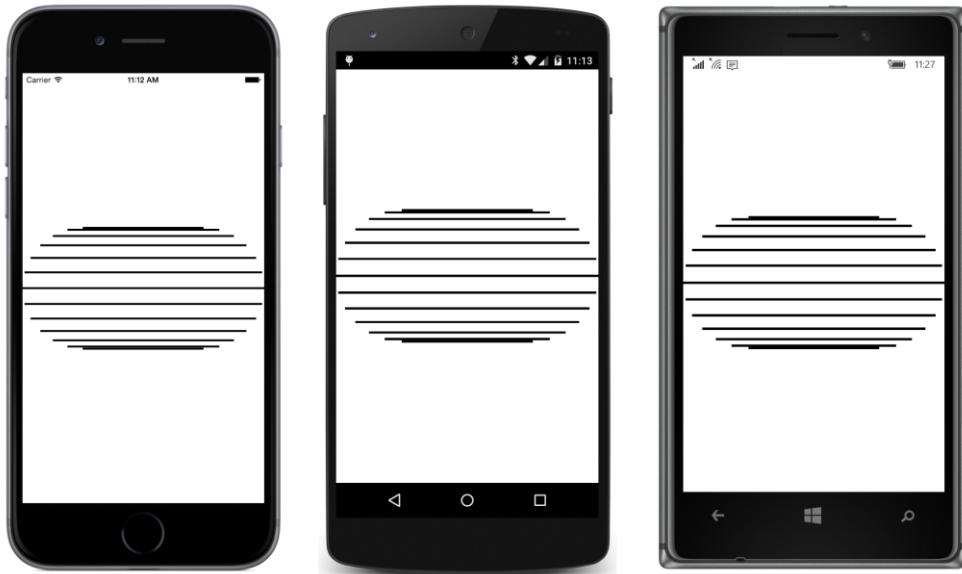
        // Buscar el centro y un tamaño para el BoxView.
        Punto centro = nuevo Punto (Dimensión / 2, dimensión / 2);
        tamaño boxViewSize = nuevo tamaño (Dimensión / 2, 3);

        para ( En t i = 0; i <numSpokes; i++)
        {
            // encontrar un ángulo para cada radio.
            doble grados = i * 360 / numSpokes;
            doble radianes = Mates .PI * grados / 180;
```

```
// encontrar el punto del centro de cada BoxView habló.  
Punto boxViewCenter =  
    nuevo Punto (Center.X + boxViewSize.Width / 2 * Matemáticos .Cos (radianes),  
    center.Y + boxViewSize.Width / 2 * Matemáticos .Sin (radianes));  
  
// Encuentra la esquina superior izquierda de la BoxView y posicionarla.  
Punto boxViewOrigin = boxViewCenter - boxViewSize * 0,5;  
AbsoluteLayout .SetLayoutBounds (boxViews [i],  
    nuevo Rectángulo (BoxViewOrigin, boxViewSize));  
  
// Girar el BoxView alrededor de su centro.  
boxViews [i] .Rotation = grados;  
}  
}  
...  
}
```

Sin duda, la manera más fácil de hacer que estos radios sería posicionar los 24 delgada BoxView elementos que se extienden hacia arriba desde el centro de la AbsoluteLayout -mucho como la posición de las 12:00 inicial de las manos de la BoxViewClock en el capítulo anterior, y después de girar cada uno de ellos alrededor de su borde inferior por un incremento de 15 grados. Sin embargo, esto requiere que el anchorY Las propiedades de estos BoxView elementos pueden ajustar a 1 para que la rotación borde inferior. Eso no funcionaría para este programa, porque cada una de las BoxView posteriormente elementos deben ser animados a girar alrededor de su centro.

La solución es calcular primero una posición dentro de la AbsoluteLayout Para el *centrar* de cada Caja-Ver. Este es el Punto valor en el SizeChanged controlador denominado boxViewCenter. los caja-ViewOrigin es entonces la esquina superior izquierda de la BoxView si el centro de la BoxView está posicionado en boxViewCenter. Si en comentario la última sentencia del para bucle que establece el Rotación propiedad de cada BoxView, verá los radios colocados de esta manera:



Todas las líneas horizontales (a excepción de la parte superior y las inferiores) son en realidad dos radios alineados. El centro de cada radio es la mitad de la longitud del radio desde el centro de la página. Rotación de cada uno de los rayos alrededor de su centro después crea el patrón inicial que vimos antes.

Aquí está la `AnimationLoop` método:

```
    público clase parcial RotatingSpokesPage : Página de contenido
    {
        ...
        vacío asíncrono AnimationLoop ()
        {
            // No te muevas durante 3 segundos.
            esperar Tarea .Delay (3000);

            // Girar la configuración de radios 3 veces.
            uint count = 3;
            esperar absoluteLayout.RotateTo (360 * recuento, 3000 * recuento);

            // Preparar para la creación de objetos de tareas.
            Lista < Tarea < bool >> Lista de tareas = nuevo Lista < Tarea < bool >> (numSpokes + 1);

            mientras ( cierto )
            {
                para cada ( BoxView boxView en boxViews )
                {
                    // Tarea para rotar cada radio.
                    taskList.Add (boxView.RelRotateTo (360, 3000));
                }

                // de tareas para rotar toda la configuración.
                taskList.Add (absoluteLayout.RelRotateTo (360, 3000));
            }
        }
    }
```

```
// Ejecutar todas las animaciones; continuará en 3 segundos  
esperar Tarea .WhenAll (lista de tareas);  
  
// Borrar la lista.  
taskList.Clear ();  
  
}  
}  
}
```

Después de la rotación preliminar de sólo el AbsoluteLayout en sí, el mientras bloque se ejecuta siempre en rotación tanto los radios y la AbsoluteLayout. Tenga en cuenta que una Lista <Tarea <bool>> se crea para almacenar 25 tareas simultáneas. Los para cada agrega un bucle Tarea a esto Lista que las llamadas RelRotateTo para cada BoxView para girar el radio de 360 grados durante tres segundos. El final Tarea es otro RelRotateTo sobre el AbsoluteLayout sí mismo.

Cuando usas RelRotateTo en una animación que se ejecuta siempre, el objetivo Rotación la propiedad se pone cada vez más y más grande y más grande. El ángulo de rotación real es el valor de la Rotación propiedad de módulo 360.

Es el valor cada vez mayor de la Rotación propiedad de un problema potencial?

En teoría, no. Incluso si la plataforma subyacente utilizado de precisión simple número de coma flotante para representar Rotación valores, un problema no surgirían hasta que el valor supera $3,4 \times 10^{38}$. Incluso si usted está aumentando la Rotación propiedad de 360 grados por segundo, y se inició la animación en el momento del Big Bang (hace 13,8 mil millones de años), la Rotación valor sería de sólo el $4,4 \times 10^{17}$.

Sin embargo, en realidad, un problema puede arrastrarse para arriba, y mucho antes de lo que se podría pensar. UN Rotación ángulo de 36.000.000-sólo 100.000 rotaciones de 360 grados-hace que un objeto que pasarán a ser un poco diferente que una Rotación ángulo de 0, y la desviación se hace más grande para una mayor Rotación ángulos.

Si desea explorar esta, encontrará un programa llamado **RotationBreakdown** entre el código fuente de este capítulo. El programa gira de dos BoxView elementos al mismo ritmo, con una RotateTo de 0 a 360 grados, y el otro con RelRotateTo con un argumento de 36000. El BoxView giradas con RotateTo normalmente oscurece la BoxView giradas con RelRotateTo, pero que por debajo BoxView es de color rojo, y en un minuto usted comenzará a ver la roja BoxView mirar a través. La desviación se hace mayor cuanto más tiempo se ejecuta el programa.

A menudo, cuando se está combinando animaciones, desea que todos ellos comienzan y terminan al mismo tiempo. Pero otras veces, y en particular con las animaciones que se ejecutan siempre, desea que varias animaciones para ejecutar de forma independiente el uno del otro, o al menos aparente para funcionar de forma independiente.

Este es el caso de la **SpinningImage** programa. El programa muestra un mapa de bits usando el **Imagen** elemento:

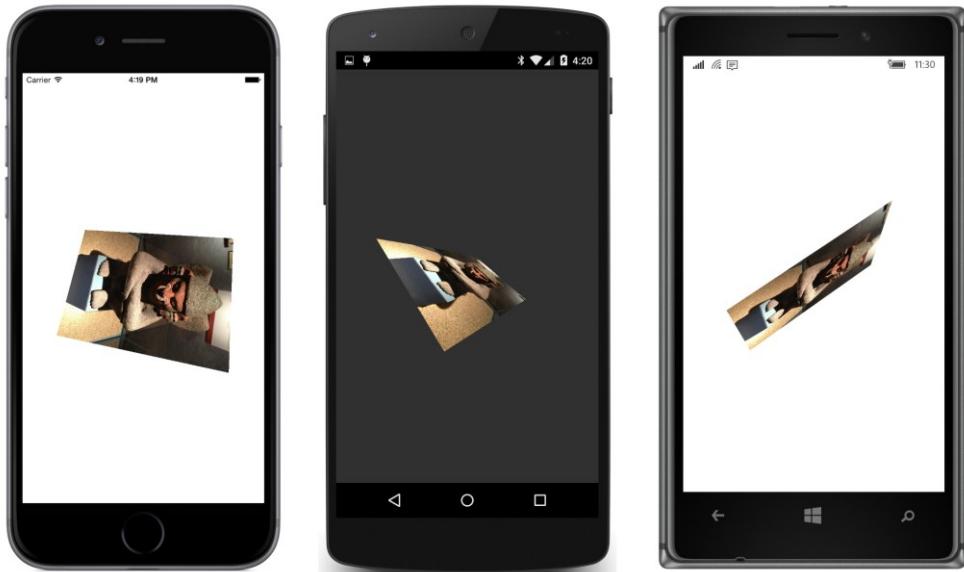
```
<Pagina de contenido xmlns = "http://xamarin.com/schemas/2014/forms"
    xmlns:x = "http://schemas.microsoft.com/winfx/2009/xaml"
    x:Class = "SpinningImage.SpinningImagePage" >
```

```
<Imagen x: Nombre = "Imagen" 
Fuente = "https://developer.xamarin.com/demo/IMG_0563.JPG" 
Escala = "0.5" />

</Pagina de contenido >
```

Normalmente, el Imagen haría que el mapa de bits para encajar en la pantalla mientras se mantiene la relación de aspecto del mapa de bits. En el modo vertical, el ancho del mapa de bits prestados sería la misma que la anchura del teléfono. Sin embargo, con una Escala ajuste de 0,5, el Imagen es la mitad de ese tamaño.

El archivo de código subyacente a continuación, lo anima mediante el uso de RotateTo, RotateXTo, y RotateYTo para hacerla girar y girar casi al azar en el espacio:



Sin embargo, es probable que no desea que el RotateTo, RotateXTo, y RotateYTo para ser sincronizado de ninguna manera porque eso daría lugar a patrones repetitivos.

La solución aquí hace realmente crear un patrón repetitivo, pero que es de cinco minutos de duración. Esta es la duración de las tres animaciones en el Task.WhenAll método:

```
público clase parcial SpinningImagePage : Pagina de contenido
{
    público SpinningImagePage ()
    {
        InitializeComponent ();

        AnimationLoop ();
    }

    vacío asíncrono AnimationLoop ()
    {
```

```
uint duración = 5 * 60 * 1000; // 5 minutos

mientras ( cierto )
{
    esperar Tarea .Cuando todo(
        image.RotateTo (307 * 360, duración),
        image.RotateXTo (251 * 360, duración),
        image.RotateYTo (199 * 360, duración));

    image.Rotation = 0;
    image.RotationX = 0;
    image.RotationY = 0;
}
}
```

Durante este período de cinco minutos, los tres animaciones separados cada uno hace un número diferente de rotaciones de 360 grados: 307 rotaciones para RotateTo, 251 de RotateXTo, y 199 para RotateYTo. Esos son todos los números primos. Ellos no tienen factores comunes. Así que nunca durante ese periodo de cinco minutos habrá dos de estas rotaciones coinciden entre sí de la misma manera.

Hay otra manera de crear animaciones simultáneas pero autónomas, pero requiere profundizar en el sistema de animación. Eso se va a ser pronto.

La animación de la propiedad bounds

Tal vez el método de extensión más curioso de ViewExtensions clase es LayoutTo. El argumento es una Rectángulo valor, y la primera pregunta podría ser: ¿Qué es la propiedad esta animación método? La única propiedad de tipo Rectángulo definido por VisualElement es el Límites propiedad. Esta propiedad indica la posición de un elemento en relación con su padre y su tamaño, pero la propiedad es de sólo conseguir.

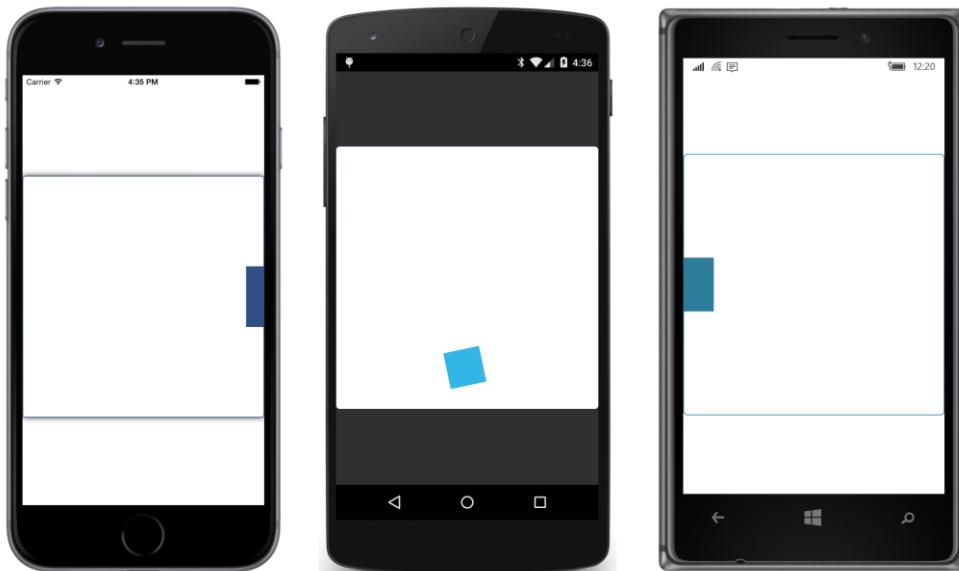
Los LayoutTo en efecto, la animación animar la Límites propiedad, pero lo hace de forma indirecta mediante una llamada al Diseño método. Los Diseño método no es algo que normalmente se llaman aplicaciones. Como su nombre indica, se utiliza comúnmente en el sistema de diseño para posicionar y niñez del tamaño en relación con sus padres. La única vez que es probable que tenga una ocasión para llamar Diseño es cuando se escribe una clase de diseño personalizado que se deriva de Disposición <Ver>, como se verá en el capítulo 26, "diseños personalizados."

Es probable que no deseas utilizar el LayoutTo animación para niños de una StackLayout o Cuadrícula porque la animación anula la posición y el tamaño establecido por el parent. Tan pronto como se enciende el teléfono de lado, la página se somete a otro pase diseño que hace que el StackLayout o Cuadrícula para mover y tamaño del niño basado en el proceso de diseño normal, y que va a anular su animación.

Vas a tener el mismo problema con un niño de una AbsoluteLayout. Después de la LayoutTo la animación completa, si a su vez el teléfono de lado, la AbsoluteLayout luego se mueve y tamaños del niño basándose en el niño de LayoutBounds adjunta propiedad enlazable. Pero con AbsoluteLayout también hay una solución a este problema: Después de la LayoutTo animación llega a la conclusión, el programa puede establecer del niño LayoutBounds adjunta propiedad que puede vincularse con el mismo rectángulo especificado en la animación, tal vez usando el ajuste final del Límites propiedad establecido por la animación.

Tenga en cuenta, sin embargo, que la **Diseño** método y la **LayoutTo** animación no tienen conocimiento del posicionamiento proporcional y característica de encolado en **AbsoluteLayout**. Si utiliza posicionamiento proporcional y dimensionamiento, puede que tenga que traducir entre las coordenadas y tamaños proporcionales y absolutos. Los **Límites** propiedad siempre los informes de posición y tamaño en coordenadas absolutas.

Los **BouncingBox** usos del programa **LayoutTo** para hacer rebotar una metódica **BoxView** por el interior de un cuadrado Marco. Los **BoxView** comienza en el centro del borde superior, luego se mueve en un arco hacia el centro del borde derecho, y luego al centro del borde inferior, el centro del borde izquierdo, y de regreso hasta la cima, desde donde el viaje continúa. A medida que la **BoxView** golpea en cada borde, de manera realista comprime y luego se expande como una pelota de goma:



Los usos de archivos de código subyacente **AbsoluteLayout.SetLayoutBounds** para posicionar el **BoxView** contra cada uno de los cuatro bordes, **LayoutTo** para la compresión y descompresión contra el borde, y **RotateTo** para mover el **BoxView** en un arco a la siguiente borde.

El archivo XAML crea la Marco, el **AbsoluteLayout**, y el **BoxView**:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns:x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " BouncingBox.BouncingBoxPage " >

< ContentPage.Padding >
    < OnPlatform x: TypeArguments = " Espesor "
        iOS = " 0, 20, 0, 0 " />
</ ContentPage.Padding >

< ContentView SizeChanged = " OnContentViewSizeChanged " >
    < Marco x: Nombre = " marco "
        OutlineColor = " Acento "
        Color de fondo = " Blanco " >
```

```

    Relleno = " 0 "
    HorizontalOptions = " Centrar "
    VerticalOptions = " Centrar " >
    <AbsoluteLayout SizeChanged = " OnAbsoluteLayoutSizeChanged " >
        <BoxView x: Nombre = " boxView "
            Color = " Acento "
            Es visible = " Falso " />
    </AbsoluteLayout >
</Marco >
</ContentView >
</Pagina de contenido >

```

En el archivo de código subyacente, la `SizeChanged` controlador para el `ContentView` ajusta el tamaño de la `Marco` a ser cuadrado, mientras que el `SizeChanged` controlador para el `AbsoluteLayout` ahorra su tamaño para los cálculos de animación y comienza la animación va si el tamaño parece ser legítimo. (Sin esta verificación, la animación comienza demasiado pronto, y utiliza un tamaño no válido de la `AbsoluteLayout`).

```

pùblico clase parcial BouncingBoxPage : Pagina de contenido
{
    estàtico uint sòlo lectura arcDuration = 1,000;
    estàtico uint sòlo lectura bounceDuration = 250;
    estàtico sòlo lectura doble boxSize = 50;
    doble layoutSize;
    bool animationGoing;

    pùblico BouncingBoxPage ()
    {
        InitializeComponent ();
    }

    vacío OnContentViewSizeChanged ( objeto remitente, EventArgs args )
    {
        ContentView contentView = ( ContentView )remitente;
        doble size = Mates .min ( contentView.Width, contentView.Height );
        frame.WidthRequest = tamaño;
        frame.HeightRequest = tamaño;
    }

    vacío OnAbsoluteLayoutSizeChanged ( objeto remitente, EventArgs args )
    {
        AbsoluteLayout AbsoluteLayout = ( AbsoluteLayout )remitente;
        layoutSize = Mates .min ( absoluteLayout.Width, absoluteLayout.Height );

        // Sólo iniciar la animación con un tamaño válido.
        Si ( ! AnimationGoing && layoutSize > 100 )
        {
            animationGoing = cierto ;
            AnimationLoop ();
        }
    }
    ...
}

```

los `AnimationLoop` método es muy largo, pero eso es sólo porque se utiliza la lógica separada para cada una de

los cuatro lados y las transiciones entre estos lados. Para cada lado, el primer paso es para posicionar el **BoxView** mediante el uso `AbsoluteLayout.SetLayoutBounds`. Entonces el **BoxView** se hace girar en un arco hacia el lado próximo. Esto requiere el establecimiento de la `anchorX` y `anchorY` propiedades para que el centro de la animación está cerca de la esquina de la Marco pero expresada en unidades de la **BoxView** tamaño.

Luego vienen las dos llamadas a `LayoutTo` para animar la compresión de la **BoxView** ya que golpea el interior de la Marco, y la posterior expansión de **BoxView** ya que rebota en:

```
público clase parcial BouncingBoxPage : Pagina de contenido
{
    ...
    vacío asíncrono AnimationLoop ()
    {
        mientras ( cierto )
        {
            // posición inicial en la parte superior.
            AbsoluteLayout .SetLayoutBounds (boxView,
                nuevo Rectángulo ((LayoutSize - boxSize) / 2, 0, boxSize, boxSize));

            // Arco de arriba a la derecha.
            boxView.AnchorX = layoutSize / 2 / boxSize;
            boxView.AnchorY = 0,5;
            esperar boxView.RotateTo (-90, arcDuration);

            // Rebote en la derecha.
            Rectángulo rectNormal = nuevo Rectángulo (LayoutSize - boxSize,
                (LayoutSize - boxSize) / 2,
                boxSize, boxSize);

            Rectángulo rectSquashed = nuevo Rectángulo (RectNormal.X + boxSize / 2,
                rectNormal.Y - boxSize / 2,
                boxSize / 2, 2 * boxSize);

            boxView.BatchBegin ();
            boxView.Rotation = 0;
            boxView.AnchorX = 0,5;
            boxView.AnchorY = 0,5;
            AbsoluteLayout .SetLayoutBounds (boxView, rectNormal);
            boxView.BatchCommit ();

            esperar boxView.LayoutTo (rectSquashed, bounceDuration, aliviar .SinOut);
            esperar boxView.LayoutTo (rectNormal, bounceDuration, aliviar .SinIn);

            // Arco de derecha a abajo.
            boxView.AnchorX = 0,5;
            boxView.AnchorY = layoutSize / 2 / boxSize;
            esperar boxView.RotateTo (-90, arcDuration);

            // rebotar en la parte inferior.
            rectNormal = nuevo Rectángulo ((LayoutSize - boxSize) / 2,
                layoutSize - boxSize,
                boxSize, boxSize);
```

```
rectSquashed = nuevo Rectángulo (RectNormal.X - boxSize / 2,
                                rectNormal.Y + boxSize / 2,
                                2 * boxSize, boxSize / 2);

boxView.BatchBegin ();
boxView.Rotation = 0;
boxView.AnchorX = 0.5;
boxView.AnchorY = 0.5;
AbsoluteLayout .SetLayoutBounds (boxView, rectNormal);
boxView.BatchCommit ();

esperar boxView.LayoutTo (rectSquashed, bounceDuration, aliviar .SinOut);
esperar boxView.LayoutTo (rectNormal, bounceDuration, aliviar .SinIn);

// Arco de abajo a la izquierda.
boxView.AnchorX = 1 - layoutSize / 2 / boxSize;
boxView.AnchorY = 0.5;
esperar boxView.RotateTo (-90, arcDuration);

// rebote a la izquierda.
rectNormal = nuevo Rectángulo (0, (layoutSize - boxSize) / 2,
                                boxSize, boxSize);

rectSquashed = nuevo Rectángulo (RectNormal.X,
                                rectNormal.Y - boxSize / 2,
                                boxSize / 2, 2 * boxSize);

boxView.BatchBegin ();
boxView.Rotation = 0;
boxView.AnchorX = 0.5;
boxView.AnchorY = 0.5;
AbsoluteLayout .SetLayoutBounds (boxView, rectNormal);
boxView.BatchCommit ();

esperar boxView.LayoutTo (rectSquashed, bounceDuration, aliviar .SinOut);
esperar boxView.LayoutTo (rectNormal, bounceDuration, aliviar .SinIn);

// Arco de izquierda a la parte superior.
boxView.AnchorX = 0.5;
boxView.AnchorY = 1 - layoutSize / 2 / boxSize;
esperar boxView.RotateTo (-90, arcDuration);

// rebotar en la parte superior.
rectNormal = nuevo Rectángulo ((LayoutSize - boxSize) / 2, 0,
                                boxSize, boxSize);

rectSquashed = nuevo Rectángulo (RectNormal.X - boxSize / 2, 0,
                                2 * boxSize, boxSize / 2);

boxView.BatchBegin ();
boxView.Rotation = 0;
boxView.AnchorX = 0.5;
boxView.AnchorY = 0.5;
AbsoluteLayout .SetLayoutBounds (boxView, rectNormal);
```

```
boxView.BatchCommit ();  
  
esperar boxView.LayoutTo (rectSquashed, bounceDuration, aliviar .SinOut);  
esperar boxView.LayoutTo (rectNormal, bounceDuration, aliviar .SinIn);  
}  
}  
}
```

los SinOut y Sinin funciones de aceleración proporcionan un poco de realismo para la compresión para reducir la velocidad, ya que está terminando, y para la expansión a acelerar después de que ha comenzado.

Note las llamadas a BatchBegin y BatchCommit que rodean una serie de valores de propiedades que acompañan a la colocación de la BoxView en uno de los bordes. Estos se añadieron porque parecía ser un poco vacilante en el simulador de iPhone, como si las propiedades no se están estableciendo al mismo tiempo. Sin embargo, el parpadeo se mantuvo incluso con estas llamadas.

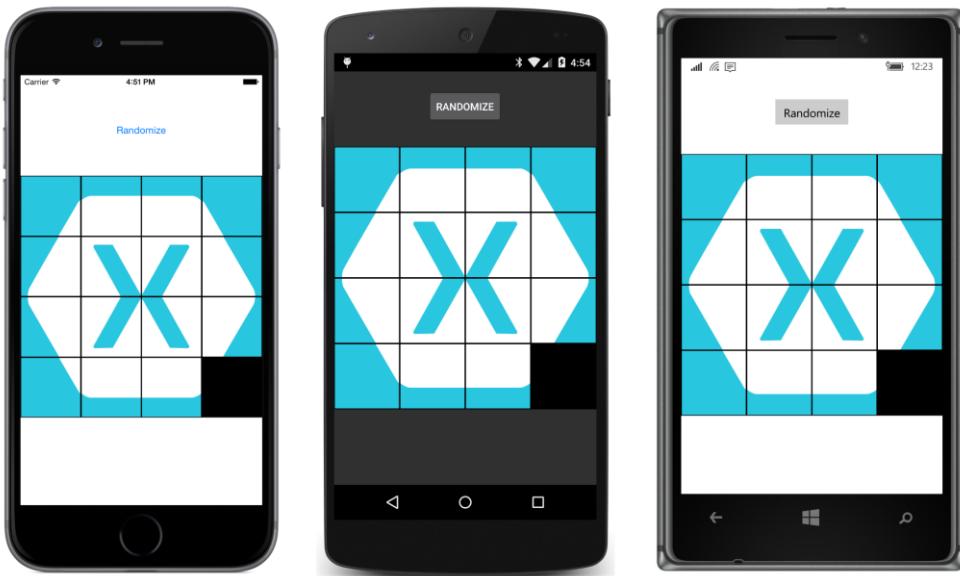
los LayoutTo animación también se utiliza en uno de los primeros juegos que fue escrito para Xamarin.Forms. Es una versión del famoso 15-rompecabezas que se compone de 15 fichas y una plaza vacía en una cuadrícula de cuatro por cuatro. Los azulejos se pueden desplazar alrededor pero sólo moviendo un azulejo en el espacio vacío.

En la temprana Apple Macintosh, este rompecabezas fue nombrado rompecabezas. En el primer kit de desarrollo de software de Windows, que era el único programa de ejemplo utilizando Microsoft Pascal, y que tenía el nombre de bozal (por "Microsoft rompecabezas"). La versión para Xamarin.Forms se llama así **Xuzzle**.

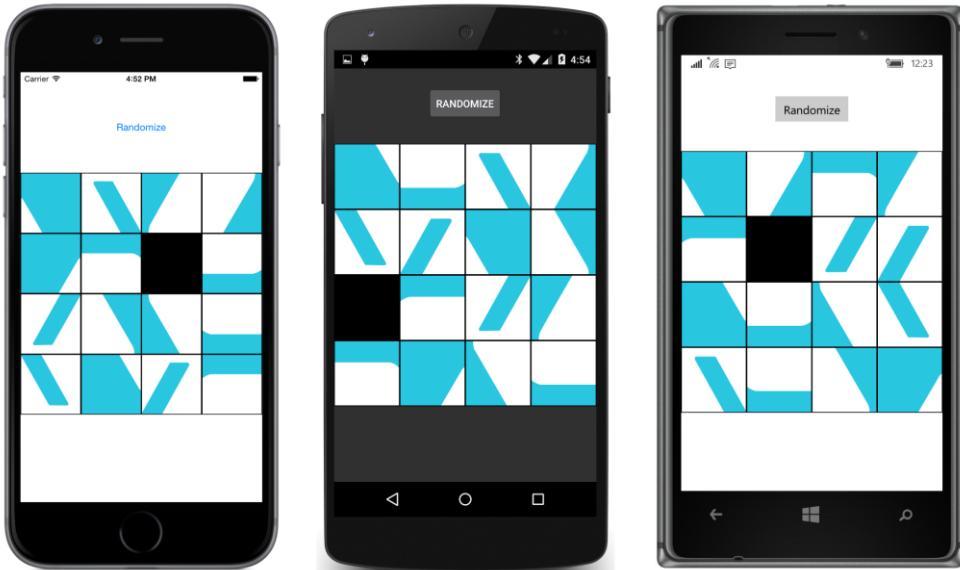
La versión original de **Xuzzle** es aquí:

<https://developer.xamarin.com/samples/xamarin-forms/Xuzzle/>

La versión algo simplificada presentada en este capítulo no incluye la animación que premios para completar con éxito el rompecabezas. Sin embargo, en lugar de mostrar letras o números, los azulejos en esta nueva visualización de la versión 15/16 del logotipo Xamarin amada, llamado el **Xamagon**, y por lo tanto esta nueva versión se llama **XamagonXuzzle**. Aquí está la pantalla de inicio:



Cuando se pulsa el aleatorizar botón, las baldosas se desplazan en torno a:



Su trabajo consiste en cambiar los azulejos de nuevo a su configuración original. Esto se hace pulsando cualquier baldosa adyacente a la plaza vacía. El programa se aplica una animación para cambiar el azulejo golpeado ligeramente en esa casilla vacía, y la plaza vacía ahora sustituye el azulejo que ha tocado.

También puede mover varias piezas con un toque. Por ejemplo, supongamos que se presiona el mosaico de más a la derecha en la tercera fila de la pantalla de Android. La segunda baldosa en esa fila mueve a la izquierda, seguido por la tercera y

cuarto azulejos también mueven a la izquierda, dejando de nuevo la plaza vacía sustitución de la teja que ha tocado.

Los mapas de bits para los 15 azulejos fueron creados especialmente para este programa, y el XamagonXuzzle proyecto los contiene en el imágenes carpeta de la biblioteca de clases portátil, todo ello con una construir Acción de Recurso incrustado.

Cada mosaico es una ContentView que simplemente contiene una Imagen con un poco Relleno aplicada a los espacios entre las baldosas que se ven en las capturas de pantalla:

```
clase XamagonXuzzleTile : ContentView
{
    público XamagonXuzzleTile ( En t file, En t columna, Fuente de imagen fuente de imagen)
    {
        Fila = fila;
        Col = col;

        Relleno = nuevo Espesor (1);
        content = nuevo Imagen
        {
            Fuente = ImageSource
        };
    }

    public int Fila { conjunto ; obtener ; }

    public int Col { conjunto ; obtener ; }
}
```

Cada mosaico tiene una fila y la columna inicial, pero el Fila y Columna propiedades son públicos, por lo que el programa puede cambiar como las baldosas se mueven alrededor. También se suministra al constructor de la XamagonXuz-zleTile clase es una Fuente de imagen objeto que hace referencia a uno de los recursos de mapa de bits.

El archivo XAML crea una instancia del Botón y un AbsoluteLayout para los azulejos:

```
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns:x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " XamagonXuzzle.XamagonXuzzlePage " >

< ContentPage.Padding >
    < OnPlatform x: TypeArguments = " Espesor "
        iOS = " 0, 20, 0, 0 " />
</ ContentPage.Padding >

< ContentView SizeChanged = " OnContentSizeChanged " >
    < StackLayout x: Nombre = " stackLayout " >
        < Botón Texto = " aleatorizar "
            hecho clic = " OnRandomizeButtonClicked "
            HorizontalOptions = " CenterAndExpand "
            VerticalOptions = " CenterAndExpand " />

        < AbsoluteLayout x: Nombre = " AbsoluteLayout "
            Color de fondo = " Negro " />

    <!-- Equilibrar diseño con botón invisible. -->
```

```

< Botón Texto = "aleatorizar"
  Opacidad = "0"
  HorizontalOptions = "CenterAndExpand"
  VerticalOptions = "CenterAndExpand" />
</ StackLayout >
</ ContentView >
</ Página de contenido >

```

Como verá, la **SizeChanged** controlador para el **ContentView** cambia la orientación de la Apilar-Diseño para dar cabida a los modos vertical y horizontal.

El constructor del archivo de código subyacente crea instancias de los 15 azulejos y da a cada uno una Fuente de imagen basado en uno de los 15 mapas de bits.

```

pública clase parcial XamagonXuzzlePage : Página de contenido
{
    // Número de azulejos horizontalmente y verticalmente,
    // pero si lo cambia, algo de código se romperá.
    estatico int sólo lectura NUM = 4;

    // Matriz de azulejos, y la fila vacía y la columna.
    XamagonXuzzleTile [] Azulejos = nuevo XamagonXuzzleTile [NUM, NUM];
    En t emptyRow = NUM - 1;
    En t emptyCol = NUM - 1;

    doble tileSize;
    bool está ocupado;

    pública XamagonXuzzlePage ()
    {
        InitializeComponent ();

        // Bucle a través de las filas y columnas.
        para ( En t fila = 0; fila <NUM; fila++)
        {
            para ( En t col = 0; Col <NUM; col++)
            {
                // Pero no tome la última!
                Si (Fila == NUM - 1 && col == NUM - 1)
                    descanso ;

                // Obtener el mapa de bits para cada baldosa y instanciarlo.
                Fuente de imagen ImageSource =
                    Fuente de imagen .FromResource ( "XamagonXuzzle.Images.Bitmap" +
                        fila + columna + ".png" );

                XamagonXuzzleTile tile = nuevo XamagonXuzzleTile (Fila, col, ImageSource);

                // Añadir el reconocimiento del grifo.
                TapGestureRecognizer tapGestureRecognizer = nuevo TapGestureRecognizer
                {
                    comando = nuevo Mando (OnTileTapped),
                    CommandParameter = baldosas
                };

```

```
tile.GestureRecognizers.Add (tapGestureRecognizer);  
  
        // Añadir a la matriz y la AbsoluteLayout.  
        azulejos [fila, col] = baldosa;  
        absoluteLayout.Children.Add (teja);  
    }  
}  
}  
...  
}
```

los SizeChanged controlador para el ContentView tiene la responsabilidad de establecer el Orientación propiedad de la StackLayout, dimensionar el AbsoluteLayout, y dimensionamiento y posicionamiento de todas las fichas dentro de la AbsoluteLayout. Observe que la posición de cada baldosa se calcula en base a la Fila y Columna propiedades de esa ficha:

El constructor ha establecido una TapGestureRecognizer en cada baldosa, y que lo maneja el OnTile-

aprovechado método. Es posible que un solo toque para resultar en hasta tres azulejos están desplazando. Ese trabajo está a cargo de la ShiftIntoEmpty método, que recorre todos los azulejos y las llamadas desplazado Animar.

Azulejo para cada uno. Ese método define la Rectángulo valor para la llamada a LayoutTo -que es el método único de animación en todo este programa, y luego otras variables se ajustan para la nueva configuración:

```
público clase parcial XamagonXuzzlePage : Pagina de contenido
{
    ...
    vacío asíncrono OnTileTapped ( objeto parámetro)
    {
        Si (está ocupado)
            regreso ;

        IsBusy = cierto ;
        XamagonXuzzleTile tappedTile = ( XamagonXuzzleTile )parámetro;
        esperar ShiftIntoEmpty (tappedTile.Row, tappedTile.Col);
        IsBusy = falso ;
    }

    asíncrono Tarea ShiftIntoEmpty ( Ent tappedRow, Ent tappedCol, uint longitud = 100)
    {
        // Desplaza columnas.
        Si (TappedRow == emptyRow && tappedCol == EmptyCol)
        {
            Ent inc = Mates .sign (tappedCol - emptyCol);
            Ent begCol = emptyCol + inc;
            Ent Columnafinal = tappedCol + inc;

            para ( Ent col = begCol; col = Columnafinal; col += inc)
            {
                esperar AnimateTile (emptyRow, col, emptyRow, emptyCol, longitud);
            }
        }
        // Desplaza filas.
        else if (TappedCol == emptyCol && tappedRow == EmptyRow)
        {
            Ent inc = Mates .sign (tappedRow - emptyRow);
            Ent begRow = emptyRow + inc;
            Ent Filafinal = tappedRow + inc;

            para ( Ent fila = begRow; remar = Filafinal; fila += inc)
            {
                esperar AnimateTile (fila, emptyCol, emptyRow, emptyCol, longitud);
            }
        }
    }

    asíncrono Tarea AnimateTile ( Ent fila, Ent columna, Ent newRow, Ent NEWCOL, uint longitud)
    {
        // El azulejo para ser animado.
        XamagonXuzzleTile animaTile = azulejos [fila, col];
    }
}
```

```

// El rectángulo de destino.
Rectángulo rect = nuevo Rectángulo (* EmptyCol tileSize,
                                         emptyRow * tileSize,
                                         tileSize,
                                         tileSize);

// animar!
esperar animaTile.LayoutTo (rect, longitud);

// Establecer los límites de diseño para mismo rectángulo.
AbsoluteLayout .SetLayoutBounds (animaTile, rect);

// Establecer varias variables y propiedades para nuevo diseño.
azulejos [newRow, Newcol] = animaTile;
animaTile.Row = newRow;
animaTile.Col = Newcol;
azulejos [fila, col] = nulo ;
emptyRow = fila;
emptyCol = col;
}
...
}

```

los AnimateTile utiliza el método esperar Para el LayoutTo llamada. Si no utilizó esperar -si se dejó que la Laico- fuera de la animación se ejecuta en segundo plano mientras se procedió con su otro trabajo, entonces el programa no sabría cuando el LayoutTo animación concluyó. Eso significa que si ShiftIntoEmpty estaban cambiando dos o tres fichas, esas animaciones ocurrirían simultáneamente en vez de secuencialmente.

Porque AnimateTile usos esperar, el método debe tener la asíncrono modificador. Sin embargo, si el método devuelve vacío, entonces el AnimateTile Método volvería cuando el LayoutTo animación comienza, y de nuevo el ShiftIntoEmpty método no sabría cuando la animación completa. Por esta razón, AnimateTile devuelve una Tarea objeto. los AnimateTile método todavía devuelve cuando el

LayoutTo animación comienza, pero devuelve una Tarea objeto que puede indicar cuando el AnimateTile Método completa. Esto significa que ShiftIntoEmpty puede llamar AnimateTile utilizando esperar y mover las fichas secuencialmente.

ShiftIntoEmpty usos esperar, por lo que también se debe definir con el asíncrono modificador, pero podría volver vacío. Si es así, entonces ShiftIntoEmpty volvería a la hora de que haga su primera llamada a AnimateTile, lo que significa que la OnTileTapped método no sabría cuando toda la animación se ha completado. Pero OnTileTapped hay que evitar que los azulejos de ser girada y animada si ya están en proceso de ser animada, la cual requiere que ShiftIntoEmpty regreso Tarea. Esto significa que OnTileTapped puedo usar esperar con ShiftIntoEmpty, Lo que significa que OnTileTapped También debe incluir la asíncrono modificador.

los OnTileTapped gestor se llama desde el Botón sí, por lo que no pueden regresar Tarea. Debe devolver vacío, al igual que el método se define. Sin embargo, se puede ver cómo el uso de esperar y asíncrono Parece creando tensión en la cadena de llamadas a métodos.

Una vez que existe el código para grifos de manejo, la aplicación de la aleatorizar botón se convierte en bastante trivial.

Simplemente hace varias llamadas a ShiftIntoEmpty con una velocidad de la animación más rápido:

```
pública clase parcial XamagonXuzzlePage : Pagina de contenido
{
    ...
    vacío asíncrono OnRandomizeButtonClicked ( objeto remitente, EventArgs args)
    {
        Botón botón = ( Botón )remitente;
        button.IsEnabled = falso ;
        Aleatorio rand = nuevo Aleatorio ();
        IsBusy = cierto ;

        // Simular algunos grifos rápido locos.
        para ( En t i = 0; i <100; i++)
        {
            esperar ShiftIntoEmpty (rand.Next (NUM), emptyCol, 25);
            esperar ShiftIntoEmpty (emptyRow, rand.Next (NUM), 25);
        }
        button.IsEnabled = cierto ;
        IsBusy = falso ;
    }
}
```

De nuevo, usando esperar con el ShiftIntoEmpty llamadas permite que las llamadas sean ejecutadas de forma secuencial (que es emocionante ver) y permite la OnRandomizeButtonClicked manejador de saber cuando todo se ha completado por lo que puede volver a activar la Botón y permitir derivaciones en las baldosas.

Sus propias animaciones awaitable

En la siguiente sección de este capítulo, verá la infraestructura subyacente de que la animación Xamarin.Forms implementos. Estos métodos subyacentes permiten definir sus propias funciones de animación que devuelven Tarea objetos y que se pueden utilizar con esperar.

En el capítulo 20, "asíncrono y el archivo de E / S," Te vi cómo utilizar la estática Task.Run método para crear un subproceso secundario de ejecución para llevar a cabo un trabajo intensivo de fondo como un cálculo de Mandelbrot. Los Task.Run método devuelve una Tarea objeto que puede indicar cuando el trabajo en segundo plano ha completado.

Pero la animación no es tan así. Una animación no tiene que pasar mucho tiempo crujir los números. Simplemente se necesita hacer algo muy breve y simple como la creación de una Rotación la propiedad una vez cada 16 milisegundos. Ese trabajo se puede ejecutar en la interfaz de usuario de hilo de hecho, el acceso a la propiedad real *debe* ejecutar en la interfaz de usuario de hilo y el tiempo puede ser manejado por el uso de Delaware-vice.StartTimer o Task.Delay.

No se debe utilizar Task.Run para la aplicación de las animaciones, debido a que una rosca secundaria de ejecución es innecesario y derrochador. Sin embargo, cuando en realidad se sienta a escribir un método de animación similar a los métodos de animación tales como Xamarin.Forms RotateTo, puede encontrarse con un obstáculo. El método debe devolver una Tarea objtar y tal vez usar Device.StartTimer por el momento, pero eso no parece posible.

He aquí un primer intento de escribir un procedimiento de este tipo. Los parámetros incluyen el objetivo VisualElement, **de y a valores, y una duración**. Usa Device.StartTimer y una Cronógrafo para calcular el valor actual de la Rotación propiedad, y que sale de la Device.StartTimer Devolución de llamada cuando la animación se ha completado:

```
Tarea MyRotate ( VisualElement visual, doble fromValue, doble valorar, uint duración )
{
    Cronógrafo cronómetro = nuevo Cronógrafo ();
    stopwatch.Start ();

    Dispositivo .StartTimer ( Espacio de tiempo .FromMilliseconds (16), () =>
    {
        doble t = Mates .min (1, stopwatch.ElapsedMilliseconds / ( doble )duración);
        doble value = fromValue + t * (toValue - fromValue);
        visual.Rotation = valor;
        bool completado = t == 1;

        Si (terminado)
        {
            // Necesidad de señalar que la tarea se ha completado. ¿Pero cómo?
        }
        regreso !terminado;
    });
}

// Necesita devolver un objeto de tareas aquí, pero ¿de dónde vienen?
}
```

En dos puntos cruciales del método no sabe qué hacer. Después de las llamadas a métodos Device.StartMinutero, que necesita para salir y regresar una Tarea oponerse a la persona que llama. Pero ¿de dónde viene esta Tarea a objetos vienen? los Tarea clase tiene un constructor, pero al igual Task.Run, que constructor crea un segundo hilo de ejecución, y no hay razón para crear ese hilo. Por otra parte, cuando la animación ha terminado, el método de alguna manera necesita señalar que la Tarea ha completado.

Afortunadamente, existe una clase que hace exactamente lo que quiere. Se llama TaskCreationSource. Es una clase genérica en el que el parámetro de tipo es el mismo que el parámetro de tipo de la Tarea objeto que desea crear. los Tarea propiedad de la TaskCreationSource objeto proporciona el Tarea

el objeto que necesita. Esto es lo que devuelve el método asíncrono. Cuando su método ha terminado de procesar el trabajo en segundo plano, puede llamar SetResult sobre el TaskCreationSource objeto, lo que indica que el trabajo está terminado.

El seguimiento TryAwaitableAnimation programa muestra cómo utilizar TaskCreationSource en un MyRotateTo método que se llama desde el hecho clic manejador de un Botón:

```
público clase parcial TryAwaitableAnimationPage : Pagina de contenido
{
    público TryAwaitableAnimationPage ()
    {
        InitializeComponent ();
    }

    vacío asíncrono OnButtonClicked ( objeto remitente, EventArgs args)
```

```

    {
        Botón botón = ( Botón )remitente;
        uint milisegundos = UInt32 .Analizar gramaticalmente( cuerda )Button.StyleId);
        esperar MyRotate( botón, 0, 360, milisegundos);
    }

    Tarea MyRotate( VisualElement visual, doble fromValue, doble valorar, uint duración)
    {
        TaskCompletionSource < objeto > = TaskCompletionSource nuevo TaskCompletionSource < objeto > ();

        Cronógrafo cronómetro = nuevo Cronógrafo ();
        stopwatch.Start();

        Dispositivo .StartTimer( Espacio de tiempo .FromMilliseconds (16), () =>
        {
            doble t = Mates .min (1, stopwatch.ElapsedMilliseconds / ( doble )duración);
            doble value = fromValue + t * (toValue - fromValue);
            visual.Rotation = valor;
            bool completado = t == 1;

            Si (terminado)
            {
                taskCompletionSource.SetResult ( nulo );
            }
            regreso Iterminado;
        });

        regreso taskCompletionSource.Task;
    }
}

```

Note la creación de instancias de TaskCreationSource, el valor de retorno de la Tarea propiedad de ese objeto, y la llamada a SetResult dentro de Device.StartTimer devolución de llamada cuando la animación ha terminado.

No hay forma de no genérico TaskCreationSource, así que si su método sólo devuelve una Tarea objeto en lugar de una Tarea <T> objeto, se debe especificar un tipo cuando se define el TaskCreationSource ejemplo. Por convención, se puede utilizar objeto para este fin, en cuyo caso sus llamadas a métodos SetResult con un nulo argumento.

los TryAwaitableAnimation archivo XAML crea una instancia de tres Botón elementos que comparten este hecho clic entrenador de animales. Cada una de ellas define su propia duración de la animación como el styleId propiedad. (Como se recordará, styleId no se usa dentro de Xamarin.Forms y existe solamente para ser utilizado por un programador de aplicaciones como una manera conveniente para adjuntar datos arbitrarios a un elemento.)

```

< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " TryAwaitableAnimation.TryAwaitableAnimationPage " >

    < StackLayout >
        < StackLayout.Resources >
            < ResourceDictionary >
                < Estilo Tipo de objetivo = " Botón " >

```

```
<Setter Propiedad = "Texto" Valor = "Puntee en Yo!" />
<Setter Propiedad = "Tamaño de fuente" Valor = "Grande" />
<Setter Propiedad = "HorizontalOptions" Valor = "Centrar" />
<Setter Propiedad = "VerticalOptions" Valor = "CenterAndExpand" />
</Estilo>
</ResourceDictionary>
</StackLayout.Resources>

<Botón hecho clic = "OnButtonClicked" styleid = "5000" />

<Botón hecho clic = "OnButtonClicked" styleid = "2500" />

<Botón hecho clic = "OnButtonClicked" styleid = "1000" />
</StackLayout>
</Pagina de contenido>
```

A pesar de que cada uno de estos Botón elementos se animando sí por una llamada a MyRotate, usted puede tener todos los botones que hacen girar al mismo tiempo. Cada llamada a MyRotate recibe su propio conjunto de variables locales, y estas variables locales se utilizan en cada Device.StartTimer llamar de vuelta.

Sin embargo, si toca una Botón mientras aún está girando, a continuación, una segunda animación se aplica a ese Botón y los dos animaciones luchan entre sí. Lo que el código requiere es una manera de cancelar la animación anterior cuando se aplica una nueva animación.

Un enfoque es para el MyRotate método para mantener un diccionario de tipo Dictionary<VisualElement, bool> se define como un campo. Siempre que ésta se una animación, MyRotate agrega el blanco VisualElement como una clave para este diccionario con un valor de falso. Cuando termina la animación, se elimina la entrada del diccionario. Un método independiente (denominada CancelMyRotate, tal vez) puede establecer el valor en el diccionario para cierto, es decir, para cancelar la animación. Los Device.StartTimer devolución de llamada puede comenzar comprobando el valor del diccionario para el particular, VisualElement y volver falso de la devolución de llamada si la animación se ha cancelado. Pero descubrirá en la discusión que sigue cómo hacerlo con menos código.

Ahora que usted ha visto las funciones de animación de alto nivel realizadas en el ViewExtensions clase, vamos a explorar cómo el resto del sistema de animación Xamarin.Forms implementa estas funciones y le permite iniciar, controlar y cancelar animaciones.

Más profundamente en la animación

En el primer encuentro, el sistema de animación Xamarin.Forms completa puede ser un poco confuso. Vamos a comenzar con una visión global de las tres clases públicas que se pueden utilizar para definir animaciones.

La clasificación de las clases

Además de aliviar clase, el sistema de animación Xamarin.Forms comprende tres clases públicas. Aquí están en orden jerárquico de nivel alto a nivel bajo:

clase ViewExtensions

Esta es la clase que ya ha visto. `ViewExtensions` es una clase estática que contiene varios métodos de extensión para `VisualElement`, que es la clase padre de `Ver` y `Página`:

- Traducir a anima el `TranslationX` y `TranslationY` propiedades
- `scaleTo` anima el Escala propiedad
- `RelScaleTo` se aplica un incremento adicional, animada o disminución de la Escala propiedad
- `RotateTo` anima el Rotación propiedad
- `RelRotateTo` se aplica un incremento adicional, animada o disminución de la Rotación propiedad
- `RotateXTo` anima el `rotationX` propiedad
- `RotateYTo` anima el `rotationY` propiedad
- Desvanecerse hacia anima el Opacidad propiedad
- `LayoutTo` anima a la de sólo conseguir Límites la propiedad llamando al Diseño método

Como se puede ver, los primeros siete métodos se dirigen a propiedades de transformación. Estas propiedades no causan ningún cambio en la forma del elemento se percibe en la disposición. Aunque la vista animada puede mover, cambiar el tamaño y rotar, ninguno de los otros puntos de vista sobre la página se ven afectados, excepto posiblemente ser oscurecida por la nueva ubicación o el tamaño.

los Desvanecerse hacia Sólo cambia la animación Opacidad propiedad, por lo que no causa cambios de diseño tampoco.

Como hemos visto, la `LayoutTo` animación es un poco diferente. El argumento es una Rectángulo valor, y el método anula esencialmente la ubicación y el tamaño asignado a la vista por el parent del elemento Diseño o Disposición <T> objeto. `LayoutTo` es más útil para la animación de los niños de una Absoluto-Diseño porque se puede llamar `AbsoluteLayout.SetLayoutBounds` con el mismo Rectángulo objeto después de la animación se ha completado. En el capítulo 26, usted aprenderá cómo utilizar `LayoutTo` en una clase que deriva de Disposición <Ver>.

Estos son todos los métodos que devuelven asíncronos Tarea <bool>. El valor de retorno es de Boole cierto si la animación fue cancelado y falso si funcionara hasta su finalización.

En adición, `ViewExtensions` También contiene una estática `ViewExtensions.CancelAnimations` método (no un método de extensión) que tiene un solo argumento de tipo `VisualElement`. Este método cancela cualquier y todas las animaciones se inició con esta clase en ese `VisualElement` objeto.

Todos los métodos de extensión en `ViewExtensions` trabajar por la creación de una o varias Animación objetos y luego llamar a la `Cometer` método definido por que `Animación` clase.

La clase de Animación

los Animación clase tiene dos constructores: un constructor sin parámetros y otro con cinco parámetros, aunque sólo se requiere uno de los argumentos:

```
Animación pública (Acción <double> devolución de llamada,
    doble inicio = 0.0f,
    doble final = 1.0f,
    Aliviar aliviando = null,
    Acción terminado = null)
```

Este define una animación de una doble valor que comienza en comienzo y termina en fin. A menudo, estos dos argumentos tendrán sus valores por defecto de 0 y 1, respectivamente. El valor animada se pasa al método de devolución de llamada como un argumento, donde generalmente se denomina t o Progreso. La devolución de llamada se puede hacer lo que quiera con este valor, pero por lo general se utiliza para cambiar un valor de una propiedad. Si la propiedad de destino es de tipo doble, entonces comienzo y fin Los valores se pueden definir los valores de inicio y fin de la propiedad animada directamente.

Animación implementa el I Enumerable interfaz. Se puede mantener una colección de animaciones niño que luego pueden ser uniformemente comenzaron y permanecen sincronizados. Para permitir que un programa para añadir elementos a esta colección, Animación define cuatro métodos:

- Añadir
- Insertar
- WithConcurrent (dos versiones)

Todos estos son fundamentalmente el mismo en que se suman a un niño Animación oponerse a una colección interna mantenida por Animación. Vas a ver ejemplos en breve.

A partir de la animación (que podría o no incluir animaciones infantiles) requiere una llamada al Cometer método. los Cometer método especifica la duración de la animación y también incluye dos devoluciones de llamada más:

```
animation.Commit (propietario IAnimatable,
    nombre de la cadena,
    tasa uint = 16,
    longitud uint = 250,
    Aliviar aliviando = null,
    Acción <doble, bool> terminó = null,
    Func <bool> repetición = null);
```

Observe que el primer argumento es IAnimatable. los IAnimatable interfaz define sólo dos métodos, denominados BatchBegin y BatchCommit. La única clase que implementa IAnimatable es VisualElement, que es la clase asociada con el ViewExtensions métodos.

los nombre argumento identifica la animación. Puede utilizar los métodos de la AnimationExtensions clase (que viene) para determinar si una animación de este nombre está en marcha o para cancelarla. No es necesario

utilizar nombres únicos para cada animación que se está ejecutando, pero si usted está haciendo la superposición de múltiples Cometer pide a la misma objeto visual, a continuación, los nombres deben ser únicos.

En teoría, el tarifa argumento indica el número de milisegundos entre cada llamada al método de devolución de llamada definido en el Animación constructor. Se ha fijado en 16 para una velocidad de la animación de 60 fotogramas por segundo, pero cambiando no tiene ningún efecto.

los repetir devolución de llamada permite que la animación se repita. Se llama al final de la animación, y si los retornos de devolución de llamada cierto, que indica que la animación se debe repetir. Como verá, funciona en algunas configuraciones pero no otros.

los Cometer método en el Animación clase funciona llamando a una Animar método en el animationExtensions clase.

clase AnimationExtensions

Me gusta ViewExtensions, AnimationExtensions es una clase estática que contiene en su mayoría los métodos de extensión. Pero mientras que el primer parámetro de la ViewExtensions métodos es una VisualElement, el primer parámetro de la AnimationExtensions métodos es una IAnimatable para ser coherente con el Cometer método en el Animación clase.

AnimationExtensions define varias sobrecargas de la Animar método con devoluciones de llamada y otra información. La versión más extensa de Animar es este método genérico:

```
public static void Animate <T> (esta auto IAnimatable,  
                                nombre de la cadena,  
                                Func <doble, T> transformar,  
                                Acción <T> de devolución de llamada,  
                                tasa uint = 16,  
                                longitud uint = 250,  
                                Aliviar aliviando = null,  
                                Acción <T, bool> terminó = null,  
                                Func <bool> repetición = null);
```

En cierto sentido, este es el único método de animación que necesita. Por ahora muchos de estos parámetros debe ser reconocible. Notar la transformar método que puede ayudar a estructurar la lógica de animaciones que se dirigen a propiedades que no son de tipo doble.

Por ejemplo, supongamos que desea animar una propiedad de tipo Color. En primer lugar, escribir un poco transformar método que acepta una doble argumento que va de 0 a 1 (y, a menudo nombrada t o Progreso) y devuelve una Color valor correspondiente a ese valor. los llamar de vuelta método obtiene que Color valor y puede luego ponerlo en una propiedad particular de un objeto en particular. Verá esta aplicación precisa al final de este capítulo.

Otros métodos públicos en el AnimationExtensions clase son AnimationIsRunning para determinar si una animación particular, en un determinado VisualElement instancia se está ejecutando, y AbortAnimation para cancelar una animación. Ambos son métodos de extensión para IAnimatable y requieren un nombre consistente con el nombre pasado a la Animar método o la Cometer método de Animación.

Trabajar con la clase de Animación

Vamos a experimentar un poco con el `Animación` clase. Esto implica objetos instanciar de tipo `Animación` y luego llamar `Comenzar`, que en realidad se inicia la animación va. Los `Comenzar` método no devuelve una `Tarea` objeto; en cambio, el `Animación` clase proporciona notificaciones enteramente a través de devoluciones de llamada.

Hay varias maneras diferentes de configurar una `Animación` objeto, y algunos de ellos podrían implicar animaciones infantiles, por lo que el proyecto que demuestra la `Animación` clase se llama **ConcurrentAnimations**. Pero no todas las manifestaciones de este programa incluyen animaciones infantiles.

El archivo XAML define sobre todo un montón de botones que sirven tanto para activar las animaciones y ser el blanco de estas animaciones:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " ConcurrentAnimations.ConcurrentAnimationsPage " >

< StackLayout >
    < StackLayout.Resources >
        < ResourceDictionary >
            < Estilo Tipo de objetivo = " Botón " >
                < Setter Propiedad = " HorizontalOptions " Valor = " Centrar " />
                < Setter Propiedad = " VerticalOptions " Valor = " CenterAndExpand " />
            </ Estilo >
        </ ResourceDictionary >
    </ StackLayout.Resources >

    < Botón Texto = " Animación 1 (Escala) "
        hecho clic = " OnButton1Clicked " />

    < Botón Texto = " Animación 2 (repetida) "
        hecho clic = " OnButton2Clicked " />

    < Botón Texto = " Detener Animación 2 "
        hecho clic = " OnStop2Clicked " />

    < Botón Texto = " Animación 3 (escala hasta &ero; abajo) "
        hecho clic = " OnButton3Clicked " />

    < Botón Texto = " Animación 4 (Escala &ero; Girar) "
        hecho clic = " OnButton4Clicked " />

    < Botón Texto = " Animación 5 (puntos) "
        hecho clic = " OnButton5Clicked " />

    < Etiqueta x: Nombre = " waitLabel "
        Tamaño de fuente = " Grande "
        WidthRequest = " 100 " />

    < Botón Texto = " Apague puntos "
        hecho clic = " OnTurnOffButtonClicked " />

    < Botón Texto = " Animación 6 (color) "
        hecho clic = " OnButton6Clicked " />
```

```
</ StackLayout >
</ Pagina de contenido >
```

El archivo de código subyacente contiene los controladores de eventos para cada uno de estos botones.

El código en el hecho clic controlador para el primer Botón utiliza los comentarios de identificar todos los argumentos a favor de la Animación constructor y el Cometer llamada. Hay un total de cuatro métodos de devolución de llamada, cada uno de los cuales se expresan aquí como una función lambda, pero no con la sintaxis más concisa:

```
público clase parcial ConcurrentAnimationsPage : Pagina de contenido
{
    ...
    público ConcurrentAnimationsPage ()
    {
        InitializeComponent ();
    }

    vacío OnButton1Clicked ( objeto remitente, EventArgs args )
    {
        Botón botón = ( Botón )remitente;

        Animación animación = nuevo Animación (
            ( doble valor) =>
            {
                button.Scale = valor;
            },
            1, // comienzo
            5, // fin
            aliviar .Lineal, // facilitando
            () =>
            {
                Depurar .Línea de escritura( "terminado" );
            }
        );
        animación.Commit (
            esta , // propietario
            "Animation1" , // nombre
            dieciséis, // tasa (pero no tiene efecto aquí)
            1000, // longitud (en milisegundos)
            aliviar .Lineal,
            ( doble valor final, bool wasCancelled) =>
            {
                Depurar .Línea de escritura( "Terminado: {0} {1}" , FinalValue, wasCancelled);
                button.Scale = 1;
            },
            () =>
            {
                Depurar .Línea de escritura( "repetir" );
                falso retorno ;
            }
        );
    }
}
```

```
...  
}
```

La devolución de llamada en el Animación constructor establece la Escala propiedad de la Botón al valor pasado a la devolución de llamada. Este valor varía de 1 a 5 como los próximos dos argumentos indican.

los Cometer método asigna un propietario a la animación. Esto puede ser el elemento visual en la que se aplica la animación u otro elemento visual, tal como la página. Los nombre se combina con el propietario para identificar de forma exclusiva la animación si debe ser cancelada. El mismo propietario se debe utilizar para las llamadas a AnimationIsRunning o AbortAnimation en el AnimationExtensions clase. (Usted verá cómo cancelar una animación en breve).

El último argumento de la Animación constructor se llama terminado, y es una devolución de llamada que se supone que es invocado cuando la animación completa, pero en esta configuración no se llama. Afortunadamente, la Cometer método también tiene una terminado devolución de llamada con dos argumentos. La primera debe indicar un valor final (pero en esta configuración que valor es siempre 1), y el segundo argumento es una bool que se establece en cierto Si se canceló la animación.

En este ejemplo, tanto terminado devoluciones de llamada hacen llamadas a Debug.WriteLine para que pueda confirmar que uno se llama pero no el otro. Los terminado devolución de llamada incluido en el Cometer establece el llamado Escala propiedad a 1, por lo que la Botón vuelve a su tamaño original.

Si desea aplicar una función de aceleración, se puede especificar ya sea en el constructor o en el Cometer llamada al método.

Los hecho clic controlador para el segundo Botón es muy similar a la primera, excepto que la sintaxis es mucho más concisa. Muchos de los parámetros al constructor y el Cometer método tienen valores por defecto, y el constructor ha tomado ventaja de los. La sintaxis de las funciones lambda también se ha simplificado:

```
pública clase parcial ConcurrentAnimationsPage : Pagina de contenido  
{  
...  
    vacío OnButton2Clicked ( objeto remitente, EventArgs args)  
    {  
        Botón botón = ( Botón )remitente;  
  
        Animación animación = nuevo Animación ( V => button.Scale = v, 1, 5);  
        animation.Commit ( esta , "Animation2" , 16, 1000, aliviar .Lineal,  
                        (V, c) => button.Scale = 1,  
                        () => cierto );  
    }  
  
    vacío OnStop2Clicked ( objeto remitente, EventArgs args)  
    {  
        esta .AbortAnimation ( "Animation2" );  
    }  
...  
}
```

La única diferencia funcional entre el código de esta Botón y el anterior Botón implica la repetir llamar de vuelta. Cuando la animación completa, es decir, después de un valor de 5 se pasa a la llamar de vuelta método de tanto el repetir y terminado devoluciones de llamada pasan a la Cometer método se llama. Si repetir devoluciones cierto, a continuación, la animación comienza de nuevo desde el principio y en el final de eso, repetir y terminado son llamados de nuevo.

Afortunadamente, el archivo XAML incluye otro Botón que las llamadas AbortAnimation poner fin a la animación. AbortAnimation es un método de extensión, por lo que debe ser llamado en el mismo elemento pasado como primer argumento a la Cometer método, que en este caso es el objeto de página.

Si desea que varias animaciones para siempre concurrentes que se ejecutan de forma independiente el uno del otro, se puede crear una Animación objeto para cada uno de ellos y luego llamar Cometer en cada uno con una repetir devolución de llamada que devuelve cierto.

animaciones infantiles

Esos dos primeros ejemplos en **ConcurrentAnimations** son animaciones individuales. Los Animación clase también soporta animaciones infantiles, y eso es lo que el controlador para el Botón la etiqueta “Animación 3” demuestra. Se crea primero un padre Animación objeto con el constructor sin parámetros. A continuación, crea dos adicionales Animación Los objetos y los añade a la matriz Animación objeto con el Añadir y

Insertar métodos:

```
público clase parcial ConcurrentAnimationsPage : Página de contenido
{
    ...
    vacío OnButton3Clicked ( objeto remitente, EventArgs args )
    {
        Botón botón = ( Botón )remitente;

        // Crear matriz objeto de animación.
        Animación parentAnimation = nuevo Animación();

        // Crear "arriba" y añadir animación a los padres.
        Animación upAnimation = nuevo Animación (
            v => button.Scale = v,
            15, aliviar .SpringIn,
            () => Depurar .Línea de escritura( "Hasta terminó" ));

        parentAnimation.Add (0, 0.5, upAnimation);

        // crear "abajo" animación y añadir a los padres.
        Animación downAnimation = nuevo Animación (
            v => button.Scale = v,
            5, 1, aliviar .Brotará,
            () => Depurar .Línea de escritura( "Abajo terminado" ));

        parentAnimation.Insert (0.5, 1, downAnimation);

        // la confirmación de animación de los padres.
        parentAnimation.Commit (
```

```

esta , "Animation3" , 16, 5000, nulo ,
(V, c) => Depurar .Línea de escritura( "Padre terminó: {0} {1}" , V, c));
}
...
}

```

Estas Añadir y Insertar métodos son básicamente los mismos, y en el uso práctico son intercambiables. La única diferencia es que Insertar devuelve la matriz Animación objeto mientras Añadir no.

Ambos métodos requieren dos argumentos de tipo doble con los nombres beginAt y finishAt. Estos dos argumentos deben estar entre 0 y 1, y finishAt debe ser mayor que beginAt. Estos dos argumentos indican el período de relativa dentro de la animación total, que estas animaciones niño en particular son activos.

La animación total es de cinco segundos de duración. Ese es el argumento de 5000 en el Cometer método. La primera animación infantil anima el Escala propiedad de 1 a 5. El beginAt y finishAt argumentos son 0 y 0,5, respectivamente, lo que significa que esta animación niño es activo durante la primera mitad de la animación-que, en general es, durante los primeros 2,5 segundos. La segunda animación niño toma la Escala propiedad del 5 de vuelta a 1. El beginAt y finishAt argumentos son 0,5 y 1, respectivamente, lo que significa que esta animación se produce en la segunda mitad de la de cinco segundos de animación en general.

El resultado es que el Botón se escala a cinco veces su tamaño más de 2,5 segundos y luego reducido a 1 durante los últimos 2,5 segundos. Notar que los dos aliviar funciones establecidas en los dos animaciones infantiles. Los Easing.SpringIn objeto hace que el Botón para reducir el tamaño inicialmente en tamaño antes de cada vez más grande, y el Easing.SpringOut función también hace que el Botón a ser más pequeños que su tamaño real hacia el final de la animación completa.

Como se verá al hacer clic en el botón para ejecutar este código, toda la terminado devoluciones de llamada se llaman ahora. Esa es una diferencia entre el uso de la Animación clase para una única animación y usarlo con animaciones infantiles. Los terminado devolución de llamada en las animaciones infantiles indica cuando ese niño en particular se ha completado, y el terminado devolución de llamada pasa al Cometer método indica cuando toda la animación ha terminado.

Hay dos diferencias más cuando se utilizan animaciones infantiles:

- Cuando se utiliza animaciones infantiles, volviendo cierto desde el repetir devolución de llamada en el Cometer método no causa la animación se repita, pero sin embargo la animación continuará funcionando sin nuevos valores.
- Si se incluye una aliviar función en el Cometer método, y el aliviar función devuelve un valor mayor que 1, la animación se dará por terminado en ese punto. Si el aliviar función devuelve un valor menor que 0, el valor se fija a igual a 0.

Si desea utilizar una aliviar función que devuelve un valor menor que 0 o mayor que 1 (por ejemplo, el Easing.SpringIn o Easing.SpringOut función), especificarlo en una o más de las animaciones infantiles, como el ejemplo demuestra, en lugar de la Cometer método.

El compilador de C# reconoce la Añadir método de una clase que implementa `IEnumerable` como un inicializador de colección. Para mantener la sintaxis de animación al mínimo, se puede seguir el nuevo operador en la matriz Animación objeto con un par de llaves para inicializar el contenido con los niños. Cada par de llaves dentro de esas llaves externas encierra los argumentos de la Añadir método. Aquí está una animación con tres hijos:

```
pública clase parcial ConcurrentAnimationsPage : Página de contenido
{
    ...
    vacío OnButton4Clicked ( objeto remitente, EventArgs args )
    {
        Botón botón = ( Botón )remitente;

        nuevo Animación
        {
            {0, 0,5, nuevo Animación ( V => button.Scale = v, 1, 5)},
            {0,25, 0,75, nuevo Animación ( V => button.Rotation = v, 0, 360)},
            {0,5, 1, nuevo Animación ( V => button.Scale = v, 5, 1)}
        }.Cometer( esta , "Animation4" , 16, 5000);
    }
    ...
}
```

Nótese también que Cometer se llama directamente sobre la Animación constructor. Esto es tan concisa como se puede hacer de este código.

Los dos primeros argumentos a estos implícita Añadir métodos indican dónde dentro de toda la animación de los padres del niño es activo. El primer hijo anima la Escala propiedad y está activo durante la primera mitad de la animación de los padres, y el último hijo también anima la Escala propiedad y está activa durante la última mitad de la animación de los padres. Ese es el mismo que el ejemplo anterior. Pero ahora también hay una animación de la Rotación propiedad con valores inicial y final de 0,25 y 0,75. Esta Rotación animación comienza a mitad de la primera Escala animación y termina a medio camino a través de la segunda Escala animación. Así es como animaciones infantiles se pueden superponer.

los Animación clase comprende también dos métodos llamados WithConcurrent añadir animaciones niño a un parent Animación objeto. Estos son similares a la Añadir y Insertar métodos, excepto que el beginAt y finishAt argumentos (o comienzo y fin como se les llama en una de las WithConcurrent métodos) no están restringidos a la gama de 0 a 1. Sin embargo, sólo la parte de la animación niño que corresponde a un rango de 0 a 1 estará activo.

Por ejemplo, supongamos que se llama WithConcurrent para definir una animación infantil que se dirige a una Escala propiedad de 1 a 4, pero con una beginAt argumento de -1 y una finishAt argumento de 2. El beginAt valor de -1 corresponde a una Escala valor de 1, y la finishAt valor de 2 corresponde a una Escala valor de 4, pero los valores fuera de la gama de 0 y 1 no juegan un papel en la animación, por lo que la Escala propiedad sólo será animado días 2 y 3.

Más allá de los métodos de animación de alto nivel

Los ejemplos de **ConcurrentAnimations** que has visto hasta ahora se han limitado a las animaciones de la Escala y Girar propiedades, por lo que no han demostrado nada que no se puede ver con los métodos de la **ViewExtensions** clase. Pero debido a que tiene acceso al método de devolución de llamada real, se puede hacer lo que quiera durante ese devolución de llamada.

He aquí una animación que se puede utilizar para indicar que su aplicación está realizando una operación que puede tardar algún tiempo en completarse. En lugar de mostrar una **ActivityIndicator**, elegido para mostrar una cadena de puntos que forma repetitiva aumenta en longitud de 0 a 10. Estos dos valores se especifican como argumentos a la **Animación** constructor. El método de devolución de llamada arroja el valor actual a un número entero para su uso con uno de los menos conocidos cuerdas constructores para construir una cadena con ese número de puntos:

```
público clase parcial ConcurrentAnimationsPage : Pagina de contenido
{
    bool keepAnimation5Running = falso ;
    ...
    vacío OnButton5Clicked ( objeto remitente, EventArgs args )
    {
        Animación animación =
            nuevo Animación (V => dotLabel.Text = nueva cadena (".", (En t) V), 0, 10);
        animation.Commit (esta , "Animation5" , 16, 3000, nulo ,
            (V, cancelado) => dotLabel.Text = "" ,
            () => KeepAnimation5Running);
        keepAnimation5Running = cierto ;
    }
    vacío OnTurnOffButtonClicked ( objeto remitente, EventArgs args )
    {
        keepAnimation5Running = falso ;
    }
    ...
}
```

los **OnButton5Clicked** método concluye estableciendo la **keepAnimation5Running** campo para cierto, y el repetir devolución de llamada en el **Cometer** método devuelve ese valor. La animación se mantendrá en funcionamiento hasta **keepAnimation5Running** se establece en falso, que es lo que el siguiente Botón hace.

La diferencia entre esta técnica y la cancelación de la animación es que esta técnica no termina inmediatamente la animación. Los repetir devolución de llamada sólo se le llama después de la animación alcanza su fin valor (que es 10 en este caso), por lo que la animación podría seguir funcionando durante casi otros tres segundos después de **keepAnimation5Running** se establece en falso.

El último ejemplo en el **ConcurrentAnimations** anima el programa de Color de fondo propiedad de la página estableciéndolo en Color valores creados por el **Color.FromHsla** método con valores de tono que van desde 0 a 1. Esta animación da el efecto de barrido a través de los colores del arco iris:

```
público clase parcial ConcurrentAnimationsPage : Pagina de contenido
```

```

{
    ...
    vacío OnButton6Clicked ( objeto remitente, EventArgs args)
    {
        nuevo Animación (Devolución de llamada: v => = BackgroundColor Color .FromHsla (v, 1, 0,5),
            empezar: 0,
            terminar: 1) .Commit (propietario: esta ,
                nombre: "Animation6",
                longitud: 5000,
                final: (v, c) => BackgroundColor = Color .Defecto);
    }
}
}

```

Este código utiliza argumentos con nombre y por lo tanto ilustra otra variación sintaxis para instanciar una Animación objeto y vocación Cometer en eso.

Más de sus propios métodos awaitable

Más temprano, que vio cómo utilizar TaskCompletionSource Juntos con Device.StartTimer para escribir sus propios métodos de animación asíncronos. También se pueden combinar TaskCompletionSource con el Animación clase para escribir es el propietario métodos de animación asíncronos similares a los de la ViewEx- tensiones clase.

Supongamos que te gusta la idea de la SlidingEntrance programa, pero no está satisfecho de que el fácil-ing.SpringOut función no funciona con el Traducir a método. Usted puede escribir su propio método de animación traducción. Si sólo necesita para animar la TranslationX propiedad, se le puede llamar TranslateXTo:

```

público estático Tarea < bool > TranslateXTo ( esta VisualElement ver, doble X,
    uint longitud = 250, aliviar aliviando = nulo )
{
    aliviando = aliviar ?? aliviar .Lineal;
    TaskCompletionSource < bool > = TaskCompletionSource nuevo TaskCompletionSource < bool > ();
    Animación animación = nuevo Animación (
        (Valor) => view.TranslationX = valor, // llamar de vuelta
        view.TranslationX, // comienzo
        X, // fin
        aliviando); // facilitando

    animation.Commit (
        ver, // propietario
        "TranslateXTo", // nombre
        dieciséis, // tarifa
        longitud, // longitud
        nulo, // facilitando
        (FinalValue, cancelado) => taskCompletionSource.SetResult (cancelado)); // terminado

    regreso taskCompletionSource.Task;
}

```

Observe que el valor actual de la TranslationX la propiedad se pasa a la Animación constructor

Para el comienzo argumento y la X parámetro para TranslateXTo se pasa como el fin argumento. Los TaskCompletionSource tiene un argumento de tipo de bool por lo que el método puede indicar si se ha cancelado o no. El método devuelve la Tarea propiedad de la TaskCompletionSource objeto y las llamadas SetResult en el terminado devolución de llamada de la Cometer método.

Sin embargo, hay un defecto sutil en este TranslateXTo método. ¿Qué ocurre si el elemento visual está animada se elimina del árbol visual durante el curso de la animación? En teoría, si no hay otras referencias a ese objeto, debe ser elegible para la recolección de basura. Sin embargo, hay **será** ser una referencia a ese objeto en el método de animación. El elemento seguirá siendo animated- e impedido de ser recogidos de la basura-a pesar de que no hay otras referencias a ese elemento!

Puede evitar esta situación particular si el método crea una animación WeakReference oponerse al elemento animado. Los WeakReference permite que el método de animación para referirse al elemento, pero no incrementa el contador de referencia para los propósitos de la recolección de basura. Si bien esto es algo que no tiene que molestar con los métodos de animación en su propia aplicación, porque usted está probablemente consciente cuando los elementos se eliminan de árboles visuales-que es algo que probablemente debería hacer en cualquier método de animación que aparece en una biblioteca.

los TranslateXTo método es en el Xamarin.FormsBook.Toolkit biblioteca, por lo que incluye el uso de WeakReference. Debido a que el elemento podría desaparecer cuando el método de devolución de llamada, el método debe obtener una referencia al elemento con el TryGetTarget método. Que devuelve el método falso si el objeto ya no está disponible:

```
espacio de nombres Xamarin.FormsBook.Toolkit
{
    público clase estática MoreViewExtensions
    {
        public static Tarea < bool > TranslateXTo (esta VisualElement ver, doble X,
                                                uint longitud = 250, aliviar aliviando = nulo )
        {
            aliviando = aliviar ?? aliviar .Lineal;
            TaskCompletionSource < bool > = TaskCompletionSource < bool > nuevo TaskCompletionSource < bool > ();
            WeakReference < VisualElement > = WeakViewRef nuevo WeakReference < VisualElement > (Vista);

            Animación animación = nuevo Animación (
                (Valor) =>
                {
                    VisualElement viewRef;
                    Si (WeakViewRef.TryGetTarget ( fuera viewRef))
                    {
                        viewRef.TranslationX = valor;
                    }
                },
                // llamar de vuelta
                view.TranslationX, // comienzo
                X, // fin
                aliviando); // facilitando

            animation.Commit (
                ver, // propietario
                ...
            );
        }
    }
}
```

```

        "TranslateXTo" ,           // nombre
        dieciséis,                // tarifa
        longitud,                 // longitud
        nulo ,                    // facilitando
        (FinalValue, cancelado) =>
            taskCompletionSource.SetResult (cancelado)); // terminado

        regreso taskCompletionSource.Task;
    }

    public static void CancelTranslateXTo ( VisualElement ver)
    {
        view.AbortAnimation ( "TranslateXTo" );
    }
    ...
}

```

Observe que un método para cancelar la animación llamado también se incluye "TranslateX".

Esta TranslateXTo método se demuestra en el SpringSlidingEntrance programa, que es el mismo que SlidingEntrance excepto que tiene una referencia a la Xamarin.FormsBook.Toolkit biblioteca y el OnAppearing llamadas de anulación TranslateXTo:

```

pública clase parcial SpringSlidingEntrancePage : Página de contenido
{
    pública SpringSlidingEntrancePage ()
    {
        InitializeComponent ();
    }

    asincrono protected void override OnAppearing ()
    {
        base .OnAppearing ();

        doble offset = 1,000;

        para cada ( Ver ver en stackLayout.Children )
        {
            view.TranslationX = offset;
            compensar * = -1;
        }

        para cada ( Ver ver en stackLayout.Children )
        {
            esperar Tarea .WhenAny (view.TranslateXTo (0, 1,000, aliviar .Brotará),
                Tarea .Delay (100));
        }
    }
}

```

La diferencia es, estoy seguro de que estaré de acuerdo, bien vale la pena el esfuerzo. Los elementos de la diapositiva en la página y la hayan rebasado los destinos antes de colocar en una página bien ordenada.

los Xamarin.FormsBook.Toolkit biblioteca también tiene una TranslateYTo método que es básicamente el

igual que TranslateXTo, pero con una sintaxis más concisa:

```
espacio de nombres Xamarin.FormsBook.Toolkit
{
    público clase estática MoreViewExtensions
    {
        ...
        public static Tarea < bool > TranslateYTo ( esta VisualElement ver, doble Y,
                                                    uint longitud = 250, aliviar aliviando = nulo )
        {
            aliviando = aliviar ?? aliviar .Lineal;
            TaskCompletionSource < bool > = TaskCompletionSource nuevo TaskCompletionSource < bool > ();
            WeakReference < VisualElement > = WeakViewRef nuevo WeakReference < VisualElement > (vista);

            Animación animación = nuevo Animación ((Valor) =>
            {
                VisualElement viewRef;
                Si (WeakViewRef.TryGetTarget ( fuera viewRef))
                {
                    viewRef.TranslationY = valor;
                }
                }, View.TranslationY, y, aliviando);

            animation.Commit (vista, "TranslateYTo" , 16, la longitud, nulo ,
                            (V, c) => taskCompletionSource.SetResult (c));

            regreso taskCompletionSource.Task;
        }

        public static void CancelTranslateYTo ( VisualElement ver)
        {
            view.AbortAnimation ("TranslateYTo");
        }
        ...
    }
}
```

Como un reemplazo para Traducir a, puedes usar TranslateXYTo. Como se vio anteriormente en este capítulo, una aliviar función que devuelve los valores inferiores a 0 o mayor que 1 no debe ser pasado a la Cometer Método para una animación con los niños. En cambio, el aliviar la función se debe pasar a la Animación constructores de los niños. Esto es lo que TranslateXYTo hace:

```
espacio de nombres Xamarin.FormsBook.Toolkit
{
    público clase estática MoreViewExtensions
    {
        ...
        public static Tarea < bool > TranslateXYTo ( esta VisualElement ver, doble X, doble Y,
                                                    uint longitud = 250, aliviar aliviando = nulo )
        {
            aliviando = aliviar ?? aliviar .Lineal;
            TaskCompletionSource < bool > = TaskCompletionSource nuevo TaskCompletionSource < bool > ();
            WeakReference < VisualElement > = WeakViewRef nuevo WeakReference < VisualElement > (vista);

            Acción < doble > CallbackX = Valor =>
```

```

    {

        VisualElement viewRef;
        Si (WeakViewRef.TryGetTarget ( fuera viewRef))
        {
            viewRef.TranslationX = valor;
        }
    };

    Acción < doble > CallbackY = Valor =>
    {
        VisualElement viewRef;
        Si (WeakViewRef.TryGetTarget ( fuera viewRef))
        {
            viewRef.TranslationY = valor;
        }
    };
};

Animación animación = nuevo Animación
{
    {0, 1, nuevo Animación (CallbackX, view.TranslationX, x, aliviando)},
    {0, 1, nuevo Animación (CallbackY, view.TranslationY, y, aliviando)}
};

animation.Commit (vista, "TranslateXYTo" , 16, la longitud, nulo ,
(V, c) => taskCompletionSource.SetResult (c));

regreso taskCompletionSource.Task;
}

public static void CancelTranslateXYTo ( VisualElement view)
{
    view.AbortAnimation ( "TranslateXYTo" );
}
...
}
}

```

La implementación de una animación de Bezier

Algunos sistemas de gráficos implementan una animación que mueve un objeto visual a lo largo de una curva de Bezier e incluso (opcionalmente) gira el objeto visual por lo que permanece tangente a la curva.

La curva de Bezier lleva el nombre de Pierre Bézier, un ingeniero y matemático francés que desarrolló el uso de la curva en diseños interactivos por computadora en carrocerías de automóvil mientras se trabaja en Renault. La curva es un tipo de spline definida por un punto de inicio y un punto final y dos puntos de control. La curva pasa por los puntos de inicio y fin pero por lo general no los dos puntos de control. En su lugar, los puntos de control funcionan como "imanes" para tirar de la curva hacia ellos.

En su forma de dos dimensiones, la curva de Bezier se representa matemáticamente como un par de ecuaciones cúbicas paramétricas.

Aquí es una BezierSpline estructura en el Xamarin.FormsBook.Toolkit biblioteca:

espacio de nombres Xamarin.FormsBook.Toolkit

```

público struct BezierSpline
{
    público BezierSpline ( Punto point0, Punto punto 1, Punto punto2, Punto punto3)
        : esta ()
    {
        Point0 = point0;
        Point1 = point1;
        Point2 = punto2;
        Point3 = POINT3;
    }

    público Punto Point0 { conjunto privado ; obtener ;}

    público Punto Punto 1 { conjunto privado ; obtener ;}

    público Punto Point2 { conjunto privado ; obtener ;}

    público Punto Point3 { conjunto privado ; obtener ;}

    público Punto GetPointAtFractionLength ( doble t, fuera Punto tangente)
    {
        // Calcular punto en la curva.

        doble x = (1 - t) * (1 - t) * (1 - t) * Point0.X +
            3 * t * (1 - t) * (1 - t) * Point1.X +
            3 * T * T * (1 - t) * Point2.X +
            t * t * t * Point3.X;

        doble y = (1 - t) * (1 - t) * (1 - t) * Point0.Y +
            3 * t * (1 - t) * (1 - t) * Point1.Y +
            3 * T * T * (1 - t) * Point2.Y +
            t * t * t * Point3.Y;

        Punto punto = nuevo Punto (X, y);

        // Calcular la tangente a la curva.

        x = 3 * (1 - t) * (1 - t) * (Point1.X - Point0.X) +
            6 * t * (1 - t) * (Point2.X - Point1.X) +
            3 * t * t * (Point3.X - Point2.X);

        y = 3 * (1 - t) * (1 - t) * (Point1.Y - Point0.Y) +
            6 * t * (1 - t) * (Point2.Y - Point1.Y) +
            3 * t * t * (Point3.Y - Point2.Y);

        tangente = nuevo Punto (X, y);

        regreso punto;
    }
}
}

```

los Point0 y Point3 puntos son los puntos de inicio y final, mientras Punto 1 y Point2 son los dos puntos de control.

los GetPointAtFractionLength método devuelve el punto de la curva correspondiente a los valores

de t que varía de 0 a 1. Los primeros cálculos de X y y en este método implicar las ecuaciones paramétricas estándar de la curva de Bezier. Cuando t es 0, el punto de la curva es Point0, y cuando t es 1, el punto de la curva es Point3.

`GetPointAtFractionLength` también tiene un segundo cálculo de X y y en base a la primera derivada de la curva, por lo que estos valores indican la tangente de la curva en ese punto. En general, pensamos en la tangente como una línea recta que toca a la curva, pero no corta, por lo que podría parecer peculiar de expresar la tangente como otro punto. Pero esto no es realmente un punto. Es un vector en la dirección desde el punto $(0, 0)$ hasta el punto (X, y) . Ese vector se puede convertir en un ángulo de rotación mediante el uso de la función tangente inversa, también conocido como el arctangente, y disponible más convenientemente a los programadores de .NET como `Math.atan2`, que tiene dos argumentos, y y X en ese orden, y devuelve un ángulo en radianes. Tendrá que convertir a grados para ajustar el `Rotación` propiedad.

El `BezierPathTo` método en el `Xamarin.Forms.Book.Toolkit` biblioteca mueve el elemento visual objetivo llamando al `Diseño` método, lo que significa que `BezierPathTo` es parecido a `LayoutTo`. El método también gira opcionalmente el elemento estableciendo su `Rotación` propiedad. En lugar de dividir el trabajo en dos animaciones infantiles, `BezierPathTo` hace todo en el método de devolución de llamada de una sola animación.

El punto de inicio de la curva de Bezier se supone que es el centro del elemento visual de que los objetivos de animación. los BezierPathTo método requiere dos puntos de control y un punto final. Todos los puntos generados a partir de la curva de Bezier también se asumen para referirse al centro del elemento visual, por lo que los puntos deben ser ajustados por el ancho y la altura media del elemento:

```

    espacio de nombres Xamarin.FormsBook.Toolkit

    {

        clase estática pública MoreViewExtensions

        {

            ...

            public static Tarea < bool > BezierPathTo (esta VisualElement ver,
                Punto pt1, Punto PT2, Punto pt3,
                uint longitud = 250,
                BezierTangent bezierTangent = BezierTangent.Ninguna,
                aliviar aliviando = nulo )

            {

                aliviando = aliviar ?? aliviar .Lineal;

                TaskCompletionSource < bool > = TaskCompletionSource nuevo TaskCompletionSource < bool > ();
                WeakReference < VisualElement > = WeakViewRef nuevo WeakReference < VisualElement > (Vista);

                Rectángulo fuera = view.Bounds;
                BezierSpline bezierSpline = nuevo BezierSpline (Bounds.Center, pt1, pt2, pt3);

                Acción < doble > Devolución de llamada = t =>

                {

                    VisualElement viewRef;
                    Si (WeakViewRef.TryGetTarget ( fuera viewRef))

                    {

                        Punto tangente;
                        Punto punto = bezierSpline.GetPointAtFractionLength (t, fuera tangente);
                        doble x = punto.x - bounds.Width / 2;
                        doble y = punto.y - bounds.Height / 2;
                    }
                }
            }
        }
    }
}

```

```

doble y = Punto.y - bounds.Height / 2;
viewRef.Layout ( nuevo Rectángulo ( nuevo Punto (X, y), bounds.Size));

Si (BezierTangent! = BezierTangent .Ninguna)
{
    viewRef.Rotation = 180 * Mates .Atan2 (tangent.Y, tangent.X) / Mates .Pi;

    Si (BezierTangent == BezierTangent .Reversed)
    {
        viewRef.Rotation += 180;
    }
}
};

Animación animación = nuevo Animación (Devolución de llamada, 0, 1, aliviando);
animation.Commit (vista, "BezierPathTo" , 16, la longitud,
final: (valor, cancelado) => taskCompletionSource.SetResult (cancelado));

regreso taskCompletionSource.Task;
}

public static void CancelBezierPathTo ( VisualElement ver)
{
    view.AbortAnimation ( "BezierPathTo" );
}
...
}

la aplicación de la Rotación ángulo es todavía un poco complicado, sin embargo. Si los puntos de una curva de Bezier se definen de manera que la curva va aproximadamente de izquierda a derecha por la pantalla, entonces la tangente es un vector que también va de izquierda a derecha, y la rotación del elemento animado debe preservar su orientación. Pero si los puntos de la curva de Bezier van de derecha a izquierda, entonces la tangente es también de derecha a izquierda, y las matemáticas dictan que el elemento debe dar la vuelta 180 grados.

```

Para controlar la orientación del elemento de destino, se define una pequeña enumeración:

```

espacio de nombres Xamarin.FormsBook.Toolkit
{
    public enum BezierTangent
    {
        Ninguna,
        Normal,
        Invertida
    }
}

```

los BezierPathTo animación utiliza esto para controlar cómo se aplica el ángulo tangente a la Rotación propiedad.

los BezierLoop programa demuestra el uso de BezierPathTo. UN Botón se encuentra en la esquina superior izquierda de una AbsoluteLayout:

```
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " BezierLoop.BezierLoopPage " >

< ContentPage.Padding >
    < OnPlatform x: TypeArguments = " Espesor "
        iOS = " 0, 20, 0, 0 " />
</ ContentPage.Padding >

< AbsoluteLayout >
    < Botón Texto = " Haga clic para Loop "
        hecho clic = " OnButtonClicked " />
</ AbsoluteLayout >
</ Pagina de contenido >
```

los hecho clic controlador para el Botón comienza por el cálculo de los puntos de inicio y final de la curva de Bezier y los dos puntos de control. El punto de partida es la esquina superior izquierda, donde la Botón inicialmente se sienta. El punto final es la esquina superior derecha. Los dos puntos de control son la esquina inferior derecha y la esquina inferior izquierda, respectivamente. Este tipo de configuración en realidad crea un bucle en la curva de Bezier:

```
público clase parcial BezierLoopPage : Pagina de contenido
{
    público BezierLoopPage ()
    {
        InitializeComponent ();
    }

    vacío asincrono OnButtonClicked ( objeto remitente, EventArgs args )
    {
        Botón botón = ( Botón ) remitente;
        Diseño matriz = ( Diseño ) Button.Parent;

        Centro // del botón en la esquina superior izquierda.
        Punto punto0 = nuevo Punto ( Button.Width / 2, button.Height / 2);

        // esquina inferior derecha de la página.
        Punto punto1 = nuevo Punto ( Parent.Width, parent.Height );

        // esquina inferior izquierda de la página.
        Punto punto2 = nuevo Punto ( 0, parent.Height );

        Centro // del botón en la esquina superior derecha.
        Punto punto3 = nuevo Punto ( Parent.Width - button.Width / 2, button.Height / 2);

        // ángulo inicial de la curva de Bezier (vector de Point0 a Point1).
        doble ángulo = 180 / Matemáticos .PI * Matemáticos .Atan2 ( point1.Y - point0.Y,
            point1.X - point0.X);

        esperar button.RotateTo ( ángulo, 1000, aliviar .SinIn);

        esperar button.BezierPathTo ( punto1, punto2, punto3, 5000,
            BezierTangent .Normal, aliviar .SinOut);

        esperar button.BezierPathTo ( punto2, point1, point0, 5000,
            BezierTangent .Reversed, aliviar .SinIn);
```

```
    esperar button.RotateTo (0, 1.000, aliviar .SinOut);  
}  
}
```

La tangente a la curva de Bezier en sus comienzos es la línea de point0 a punto 1. Este es el ángulo variable que calcula el método para que pueda utilizar primero RotateTo para hacer girar el Botón para evitar un salto cuando el BezierPathTo animación comienza. El primero BezierPathTo mueve el Botón desde la esquina superior izquierda a la esquina superior derecha con un lazo en la parte inferior de la pantalla:



Un segundo BezierPathTo a continuación, invierte el viaje de vuelta a la esquina superior izquierda. (Aquí es donde el BezierTangent enumeración entra en juego. Sin él, el Botón De repente voltearía al revés como el segundo BezierPathTo comienza.) Una última RotateTo restaura a su orientación original.

Trabajar con AnimationExtensions

Por que ViewExtensions No incluya una colorTo ¿animación? Hay tres posibles razones por las que un procedimiento de este tipo no es tan evidente como se podría suponer inicialmente:

En primer lugar, el único Color Propiedad Definido por VisualElement es Color de fondo, pero eso no es por lo general el Color propiedad que desea animar. Es más probable que desea animar el Color de texto propiedad de Etiqueta o el Color propiedad de BoxView.

En segundo lugar, todos los métodos en ViewExtensions animar una propiedad de su valor actual a un valor especificado. Pero a menudo el valor actual de una propiedad de tipo Color es Color.Default, que no es un color real y que no se puede utilizar en un cálculo de interpolación.

En tercer lugar, la interpolación entre dos Color los valores se pueden calcular en una variedad de maneras diferentes,

pero hay dos que destacan como el más probable: Es posible que desee para interpolar los valores de color rojo-verde-azul o los valores de tono-saturación-luminosidad. Los valores intermedios serán diferentes en estos dos casos.

Vamos a cuidar de estos tres problemas con tres soluciones diferentes:

En primer lugar, no vamos a tener el método de animación de color de destino una propiedad particular. Vamos a escribir el método con un método de devolución de llamada que pasa el interpolador valor de nuevo a la persona que llama.

En segundo lugar, vamos a exigir que tanto un comienzo Color valor y un fin Color suministrarse valor al método de animación.

En tercer lugar, vamos a escribir dos métodos diferentes, `RgbColorAnimation` y `HslColorAnimation`.

Por supuesto que podría utilizar el `Animation` clase y `Cometer` para este trabajo, pero en su lugar vamos a profundizar más en el sistema de animación Xamarin Forms y utilizar un método en el `AnimationExtensions` clase.

AnimationExtensions tiene cuatro métodos diferentes nombradas **Animar**, así como una **AnimateKinetic** método. El **AnimateKinetic** método está destinado a aplicar un valor de “arrastrar” a una animación para que se ralentiza como por fricción. Sin embargo, todavía no está funcionando de una manera que permita que los resultados sean fácilmente predecibles, y no se ha demostrado en este capítulo.

De los cuatro Animar métodos, la forma genérica es el más versátil:

El tipo genérico es el tipo de la propiedad que desea animar, por ejemplo, Color. Por este tiempo se debe reconocer todos estos parámetros, excepto para el método de devolución de llamada con nombre transformar. La entrada a la devolución de llamada es siempre una t o Progreso valor que va de 0 a 1. La salida es un valor del tipo por genéricos ejemplo, Color. Ese valor se pasa a continuación a la llamar de vuelta método para la aplicación a una propiedad particular.

Aquí están `RgbColorAnimation` y `HslColorAnimation` en el `MoreViewExtensions` clase del `Xamarin.FormsBook.Toolkit` biblioteca:

```
espacio de nombres Xamarin.FormsBook.Toolkit
{
    clase estática pública MoreViewExtensions
    {
        ...
        public static Tarea<bool> RgbColorAnimation (esta VisualElement ver,
            Color fromColor, Color colorear,
            Acción <Color> Devolución de llamada,
            uint longitud = 250,
```

```

        aliviar aliviando = nulo );

    {
        Func < doble , Color > Transformar = (t) =>
    {
        regreso Color .FromRgba (fromColor.R + t * (toColor.R - fromColor.R),
                                fromColor.G + t * (toColor.G - fromColor.G),
                                fromColor.B + t * (toColor.B - fromColor.B),
                                fromColor.A + t * (toColor.A - fromColor.A));
    };

        regreso ColorAnimation (vista, "RgbColorAnimation" , Transformar,
                               de devolución de llamada, la duración, aliviando);
    }

    public static void CancelRgbColorAnimation ( VisualElement ver)
    {
        view.AbortAnimation ( "RgbColorAnimation" );
    }

    public static Tarea < bool > HslColorAnimation ( esta VisualElement ver,
                                                    Color fromColor, Color colorear,
                                                    Acción < Color > Devolución de llamada,
                                                    uint longitud = 250,
                                                    aliviar aliviando = nulo )
    {

        Func < doble , Color > Transformar = (t) =>
    {
        regreso Color .FromHsla (
            fromColor.Hue + t * (toColor.Hue - fromColor.Hue),
            fromColor.Saturation + t * (toColor.Saturation - fromColor.Saturation),
            fromColor.Luminosity + t * (toColor.Luminosity - fromColor.Luminosity),
            fromColor.A + t * (toColor.A - fromColor.A));
    };

        regreso ColorAnimation (vista, "HslColorAnimation" , Transformar,
                               de devolución de llamada, la duración, aliviando);
    }

    public static void CancelHslColorAnimation ( VisualElement ver)
    {
        view.AbortAnimation ( "HslColorAnimation" );
    }

    estético Tarea < bool > ColorAnimation ( VisualElement ver,
                                              cuerda nombre,
                                              Func < doble , Color > Transformar,
                                              Acción < Color > Devolución de llamada,
                                              uint longitud,
                                              aliviar flexibilización)
    {

        aliviando = aliviar ?? aliviar .Lineal;
        TaskCompletionSource < bool > = TaskCompletionSource nuevo TaskCompletionSource < bool > ();
        view.Animate < Color > (Nombre, transformar, devolución de llamada, 16,

```

```

        longitud, facilitando, (valor, cancelado) =>
    {
        taskCompletionSource.SetResult (cancelado);
    });

    regreso taskCompletionSource.Task;
}
}
}

```

Los dos métodos definen su propia transformar funciones y luego hacen uso de lo privado ColorAnimation Método para realmente hacer la llamada a la Animate método en el AnimationExtensions.

Debido a que estos métodos no se dirigen explícitamente un elemento visual en particular, no hay ninguna necesidad de que el WeakReference clase.

los **ColorAnimations** programa demuestra estos métodos para la animación de varias propiedades de color de varias maneras. El archivo XAML como Etiqueta, dos Botón elementos, y dos BoxView elementos:

```

< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " ColorAnimations.ColorAnimationsPage " >

    < StackLayout >
        < Etiqueta x: Nombre = " etiqueta "
            Texto = " TEXTO "
            Tamaño de fuente = " 48 "
            FontAttributes = " Negrita "
            HorizontalOptions = " Centrar "
            VerticalOptions = " CenterAndExpand " />

        < Botón Texto = " Antecedentes del arco iris "
            hecho clic = " OnRainbowBackgroundColorClicked "
            HorizontalOptions = " Centrar "
            VerticalOptions = " CenterAndExpand " />

        < Botón Texto = " BoxView color "
            hecho clic = " OnBoxViewColorButtonClicked "
            HorizontalOptions = " Centrar "
            VerticalOptions = " CenterAndExpand " />

        < StackLayout Orientación = " Horizontal " >
            < BoxView x: Nombre = " boxView1 "
                Color = " Azul "
                HeightRequest = " 100 "
                HorizontalOptions = " FillAndExpand " />

            < BoxView x: Nombre = " boxView2 "
                Color = " Azul "
                HeightRequest = " 100 "
                HorizontalOptions = " FillAndExpand " />
        </ StackLayout >
    </ StackLayout >
</ Página de contenido >

```

El archivo de código subyacente utiliza una mezcla de `RgbColorAnimation` y `HslColorAnimation` para animar los colores de la Etiqueta texto y su fondo, el fondo de la página, y los dos BoxView elementos.

los Etiqueta texto y su fondo se animan continuamente de forma opuesta entre blanco y negro. Sólo a mediados del animaciones cuando tanto el texto y el fondo son gris medio-es el texto invisible:

```
pública clase parcial ColorAnimationsPage : Pagina de contenido
{
    pública ColorAnimationsPage ()
    {
        InitializeComponent ();

        AnimationLoop ();
    }

    vacío asíncrono AnimationLoop ()
    {
        mientras ( cierto )
        {
            Acción < Color > TextCallback = color => label.TextColor = color;
            Acción < Color > BackCallback = color => label.BackgroundColor = color;

            esperar Tarea .Cuando todo(
                label.RgbColorAnimation ( Color .Blanco, Color .Black, textCallback, 1000),
                label.HslColorAnimation ( Color .Negro, Color .white, backCallback, 1000));

            esperar Tarea .Cuando todo(
                label.RgbColorAnimation ( Color .Negro, Color .white, textCallback, 1000),
                label.HslColorAnimation ( Color .Blanco, Color .Black, backCallback, 1000));
        }
    }
    ...
}
```

Cuando la animación de entre De color negro y Color blanco, no importa si se utiliza `RGB-ColorAnimation` o `HslColorAnimation`. El resultado es el mismo. Negro está representado en RGB como (0, 0, 0) y en HSL como (0, 0, 0). El blanco es (1, 1, 1) en RGB y (0, 0, 1) en HSL. En el punto a mitad de camino, el color RGB (0.5, 0.5, 0.5) es el mismo que el color HSL (0, 0, 0.5).

los `HslColorAnimation` es ideal para la animación a través de todos los matices, que corresponden aproximadamente a los colores del arco iris, tradicionalmente rojo, naranja, amarillo, verde, azul, Indigo y violeta. En las animaciones de color, una animación final de nuevo a rojo por lo general se produce al final. La animación de los colores RGB a través de esta secuencia de animación requiere en primer lugar de Color rojo a Color amarillo, entonces Color amarillo a

Color verde, entonces Color verde a Color.Aqua, entonces Color.Aqua a Color azul, entonces Color azul a Color.Fuchsia, y finalmente Color.Fuchsia a Color rojo.

Con `HslColorAnimation`, todo lo que es necesario es para animar entre dos representaciones de rojo, uno con el Matiz se pone a 0 y la otra con el Matiz establecido en 1:

```

pública clase parcial ColorAnimationsPage : Pagina de contenido
{
    ...
    vacío asíncrono OnRainbowBackgroundButtonClicked ( objeto remitente, EventArgs args)
    {
        // Animación de rojo a rojo.
        esperar a este .HslColorAnimation ( Color .FromHsla (0, 1, 0,5),
            Color .FromHsla (1, 1, 0,5),
            color => BackgroundColor = color,
            10000);

        BackgroundColor = Color .Defecto;
    }
    ...
}

```

Incluso con animaciones simples entre dos colores primarios, RgbColorAnimation y HslColorAnimation puede producir resultados diferentes. Considere una animación de azul a rojo. Los **ColorAnimations** programa demuestra la diferencia por la animación de los colores de dos BoxView elementos con los dos métodos de animación:

```

pública clase parcial ColorAnimationsPage : Pagina de contenido
{
    ...
    vacío asíncrono OnBoxViewColorButtonClicked ( objeto remitente, EventArgs args)
    {
        Acción < Color > Callback1 = color => boxView1.Color = color;
        Acción < Color > Callback2 = color => boxView2.Color = color;

        esperar Tarea .WhenAll (boxView1.RgbColorAnimation ( Color .Azul, Color .red, callback1, 2000),
            boxView2.HslColorAnimation ( Color .Azul, Color .Red, callback2, 2000));

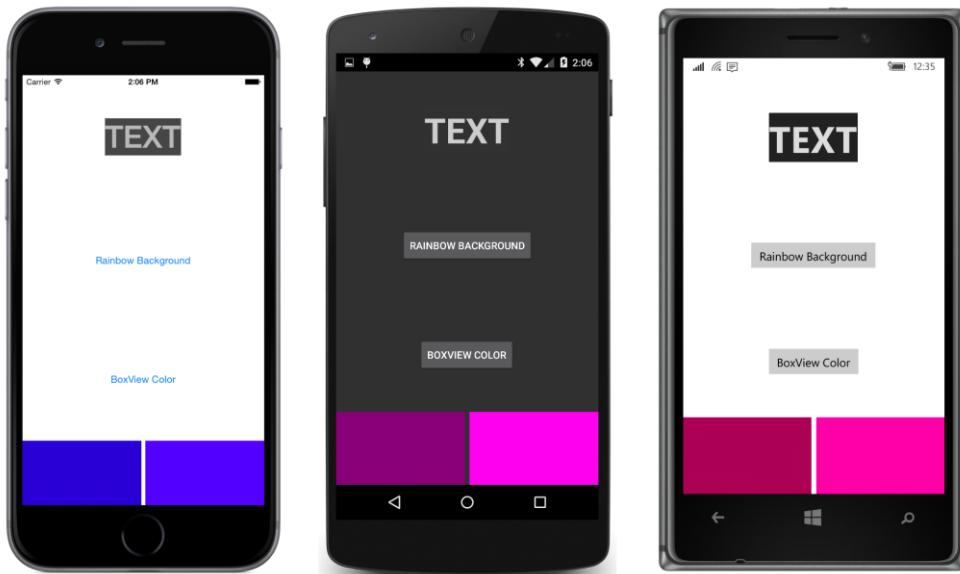
        esperar Tarea .WhenAll (boxView1.RgbColorAnimation ( Color .Rojo, Color .Blue, callback1, 2000),
            boxView2.HslColorAnimation ( Color .Rojo, Color .Blue, callback2, 2000));
    }
}

```

Azul tiene una representación RGB de (0, 0, 1) y una representación HSL de (0,67, 1, 0,5). Red tiene una representación RGB de (1, 0, 0) y en HSL es (1, 0, 0,5). A mitad de camino a través de la animación RGB, el color interpolada es (0,5, 0, 0,5), que se conoce en Xamarin.Forms como Color.Magenta. Sin embargo, a medio camino a través de la HslColorAnimation, el color interpolada es (0,83, 1, 0,5), que es el más ligero

Color.Fuchsia, que tiene una representación RGB de (1, 0, 1).

Esta captura de pantalla muestra el progreso (de izquierda a derecha) de la animación de los dos BoxView elementos de azul a rojo:



Ninguno de los dos es “correcto” o “incorrecto”. Es tan sólo dos formas diferentes de interpolación entre dos colores, y la razón por un simple `ColorAnimation` método es inadecuado.

La estructuración de sus animaciones

No hay representación XAML de animaciones, por lo que gran parte del enfoque de este capítulo ha sido necesariamente el código en lugar de marcado.

Sin embargo, cuando se utiliza en conjunción con animaciones estilos, y con MVVM y el enlace de datos, es probable que desee una manera de referirse a las animaciones en XAML. Esto es posible, y verá en el siguiente capítulo cómo se puede encapsular animaciones dentro de las clases denominadas *las acciones de activación y comportamientos*, y luego hacerlos parte del peinado y el enlace de datos de elementos visuales de la aplicación.

capítulo 23

Disparadores y comportamientos

La introducción de un lenguaje de marcado tales como XAML en un entorno de programación gráfica podría parecer a primera vista meramente una forma alternativa para la construcción de un ensamblaje de elementos de interfaz de usuario. Pero hemos visto que el lenguaje de marcado tiende a tener consecuencias mucho más profundas. El lenguaje de marcas nos induce a dividir el programa con mayor decisión entre los elementos visuales interactivos y la lógica de negocio subyacente. Esto sugiere, además, que podrían beneficiarse de la formalización de dicha separación en una arquitectura de aplicaciones tales como MVVM, y que resulta ser muy valiosa.

Al mismo tiempo, los lenguajes de marcas como XAML tienden a tener algunas deficiencias intrínsecas en comparación con el código. Mientras el código generalmente define un proceso dinámico, lenguajes de marcado suelen limitarse a la descripción de un estado fijo. Varias características se han añadido a Xamarin.Forms para ayudar a compensar estas deficiencias. Estas características incluyen extensiones de marcado, el diccionario de recursos, estilos y enlace de datos.

En este capítulo, verá dos más de estas características, llamado *disparadores y comportamientos*. Triggers causan cambios en la interfaz de usuario en respuesta a eventos o cambios de propiedad, mientras que los comportamientos son más abierta, lo que permite trozos enteros de la funcionalidad que se añade a los elementos visuales existentes. Los dos gatillos y los comportamientos pueden ser parte de una Estilo definición. A menudo, los disparadores y comportamientos son compatibles con el código que puede contener animaciones.

Es poco probable que los disparadores y comportamientos siquiera se han concebido o inventado en un entorno de programación codeonly. Sin embargo, como el diccionario de recursos, estilos, y el enlace de datos, estas características ayudan a los desarrolladores estructurar sus aplicaciones de forma más productiva, al sugerir maneras de conceptualizar las diversas piezas y componentes de estos programas y otras maneras de utilizar y compartir código.

Disparadores y comportamientos se implementan con varias clases que se introducirán en el curso de este capítulo. Usted va a hacer uso de estos disparadores y comportamientos con propiedades de dos de recolección que se definen por tanto VisualElement y Estilo:

- **disparadores propiedad de tipo IList <TriggerBase>**
- **comportamientos propiedad de tipo IList <Comportamiento>**

Vamos a empezar con los desencadenantes.

disparadores

En el más general (y más vaga) sentido, un disparo es una condición que resulta en una respuesta. Más específicamente, un disparador responde a un cambio de propiedad o el disparo de un evento mediante el establecimiento de otra propiedad o ejecutar algún código. Casi siempre, las propiedades que se establecen, o el código que se ejecuta, implican la interfaz de usuario y se representan en XAML.

Ambos VisualElement y Estilo definir una disparadores propiedad de tipo IList <TriggerBase>. El abstracto TriggerBase clase se deriva de BindableObject. Cuatro clases cerradas derivan de Desencadenar-

Base:

- Desencadenar para establecer las propiedades (o la ejecución de código) en respuesta a un cambio de propiedad.
- EventTrigger para ejecutar código en respuesta a un evento.
- DataTrigger para establecer las propiedades (o la ejecución de código) en respuesta a un cambio de propiedad hace referencia en un enlace de datos.
- MultiTrigger para establecer las propiedades (o la ejecución de código) cuando se producen múltiples factores desencadenantes.

Las diferencias entre estos se convertirán en mucho más clara en la práctica.

El gatillo simple

En su forma más habitual, el Desencadenar cheques de clase para un cambio de propiedad de un elemento y responde mediante el establecimiento de otra propiedad del mismo elemento.

Por ejemplo, supongamos que se ha diseñado una página que contiene varios Entrada puntos de vista. Usted ha decidido que cuando un particular, Entrada recibe el foco de entrada, desea que el Entrada a ser más grande. ¿Quieres hacer el Entrada destacan, incluyendo el texto que el usuario escribe.

Mucho más específicamente, cuando el Esta enfocado propiedad de la Entrada se convierte Ciento, quiere que el Escala propiedad de la Entrada a ser fijado a 1,5. Cuando el Esta enfocado propiedad revierte de nuevo a Falso, quiere que el Escala propiedad para volver también a su valor anterior.

Para adaptarse a este concepto, Desencadenar define tres propiedades:

- Propiedad de tipo BindableProperty.
- Valor de tipo Objeto.
- setters de tipo IList <Setter>. Esta es la propiedad del contenido Desencadenar.

Todas estas propiedades deben estar establecidas para el Desencadenar trabajar. De TriggerBase, Gatillo hereda otra propiedad esencial:

- Tipo de objetivo de tipo Tipo.

Este es el tipo del elemento en el que la Desencadenar se adjunta.

los Propiedad y Valor propiedades de Desencadenar A veces se dice que constituyen una *condición*.

Cuando el valor de la propiedad que hace referencia Propiedad es igual Valor, la condición es verdadera, y la colección de Setter objetos se aplican al elemento.

Como recordará del capítulo 12, "Estilos" Setter define dos propiedades que resultan ser los mismos que los dos primeros Desencadenar propiedades:

- Propiedad de tipo BindableProperty.
- Valor de tipo Objeto.

Con disparadores que sólo estamos tratando con propiedades enlazables. los Desencadenar la propiedad Estado debe estar respaldada por una BindableProperty así como la propiedad establecida por el Setter.

Cuando la condición se convierte en falsa, la Setter los objetos son "no-aplicada", lo que significa que la propiedad referencia el Setter revierte a su valor lo que sería sin la Setter, que podría ser el valor predeterminado de la propiedad, un valor fijado directamente sobre el elemento, o un valor aplicado a través de una Estilo.

Aquí está el archivo XAML para el **entrypop** programa. Cada uno de los tres Entrada vistas en la página tiene una sola Desencadenar objeto añadió a su disparadores recogida mediante el Entry.Triggers etiqueta de propiedad de elementos. Cada una de las Desencadenar objetos tiene una sola Setter añadido a su setters colección. Porque

setters es la propiedad de contenido Desencadenar, el Trigger.Setters No se requieren etiquetas de propiedad de elementos:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " EntryPop.EntryPopPage "
    Relleno = " 20, 50, 120, 0 ">

    < StackLayout Espaciado = " 20 " >
        < Entrada marcador de posición = " Ingrese su nombre "
            anchorX = " 0 " >
            < Entry.Triggers >
                < Desencadenar Tipo de objetivo = " Entrada " Propiedad = " Esta enfocado " Valor = " Cierto " >
                    < Setter Propiedad = " Escala " Valor = " 1.5 " />
                </ Desencadenar >
            </ Entry.Triggers >
        </ Entrada >

        < Entrada marcador de posición = " ingresa la dirección "
            anchorX = " 0 " >
            < Entry.Triggers >
                < Desencadenar Tipo de objetivo = " Entrada " Propiedad = " Esta enfocado " Valor = " Cierto " >
                    < Setter Propiedad = " Escala " Valor = " 1.5 " />
                </ Desencadenar >
            </ Entry.Triggers >
        </ Entrada >

        < Entrada marcador de posición = " entrar en la ciudad y el estado "
            anchorX = " 0 " >
            < Entry.Triggers >
```

```

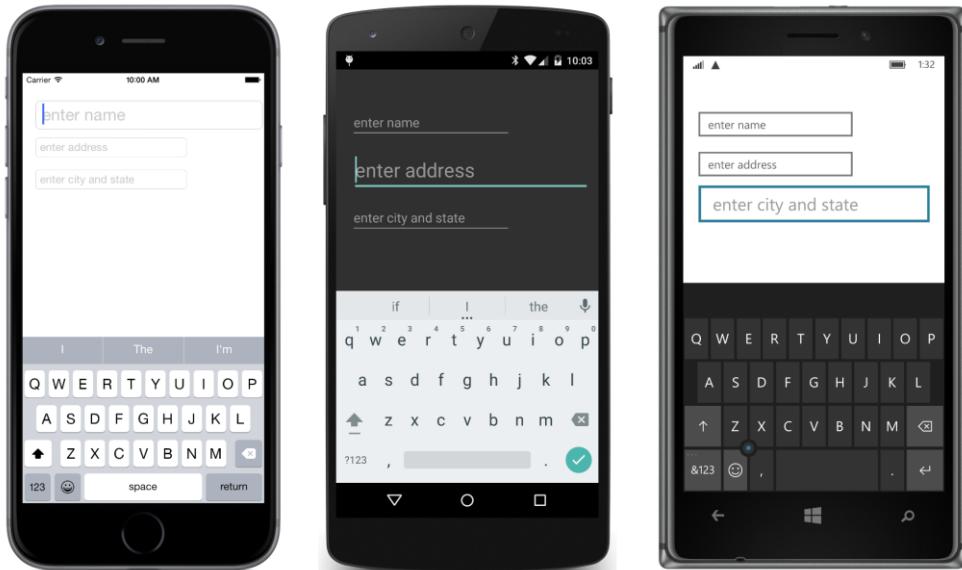
    anchorX = "0">
<Entry.Triggers>
    <Desencadenar Tipo de objetivo = "Entrada" Propiedad = "Esta enfocado" Valor = "Ciento" >
        <Setter Propiedad = "Escala" Valor = "1.5" />
    </Desencadenar>
</Entry.Triggers>
</Entrada>
</StackLayout>
</Pagina de contenido>

```

Cada Desencadenar objeto debe tener su Tipo de objetivo Establecer y, en este caso, que es una Entrada. Internamente, el Desencadenar utiliza una PropertyChanged controlador para controlar el valor de la Esta enfocado propiedad. Cuando esa propiedad es igual Ciento, entonces el único Setter objeto establece el Escala propiedad a 1,5. los anchorX ajuste de cero dirige la escala que se produzca desde el lado izquierdo de la Entrada. (El valor no predeterminado de anchorX significa que el Entrada puntos de vista no serán colocadas correctamente si cambia la orientación de la pantalla IOS).

Cuando el Entrada pierde el foco de entrada y la Esta enfocado propiedad se convierte Falso de nuevo, el Trigonometría-ger quita automáticamente la aplicación de la Setter, en cuyo caso el Escala propiedad revierte a su pre- Desencadenar valor, que no es necesariamente su valor por defecto.

Éstos son el ampliada Entrada puntos de vista con el foco de entrada:



Cada Entrada Ver en este ejemplo tiene sólo una Desencadenar, y cada Desencadenar tiene un solo Setter, pero en el caso general, un elemento visual puede tener múltiples Desencadenar objetos en su disparadores colección, y cada Desencadenar puede tener múltiples Setter objetos en su setters colección.

Si se va a hacer algo como esto en el código, que le adjunta una PropertyChanged controlador de eventos para cada uno Entrada y responder a los cambios en la Esta enfocado propiedad estableciendo la Escala propiedad. los

ventaja de la Desencadenar es que se puede hacer todo el trabajo en el derecho de marcas, donde se define el elemento, dejando código para trabajos presumiblemente más importante que aumentar el tamaño de una Entrada ¡elemento!

Por esta razón, es poco probable que va a tener la necesidad de crear Desencadenar objetos en el código. Sin embargo, el **EntryPopCode** programa demuestra cómo usted lo haría. El código ha sido formado para parecerse a la XAML tanto como sea posible:

```
clase pública EntryPopCodePage : Pagina de contenido
{
    público EntryPopCodePage ()
    {
        Relleno = nuevo Espesor (20, 50, 120, 0);
        content = nuevo StackLayout
        {
            Espaciado = 20,
            Los niños =
            {
                nuevo Entrada
                {
                    = marcador de posición "Ingrese su nombre",
                    AnchorX = 0,
                    disparadores =
                    {
                        nuevo Desencadenar ( tipo de ( Entrada ))
                        {
                            propiedad = Entrada .IsFocusedProperty,
                            valor = cierto ,
                            setters =
                            {
                                nuevo Setter
                                {
                                    propiedad = Entrada .ScaleProperty,
                                    Valor = 1,5
                                }
                            }
                        }
                    }
                },
                nuevo Entrada
                {
                    = marcador de posición "Entrar en address",
                    AnchorX = 0,
                    disparadores =
                    {
                        nuevo Desencadenar ( tipo de ( Entrada ))
                        {
                            propiedad = Entrada .IsFocusedProperty,
                            valor = cierto ,
                            setters =
                            {
                                nuevo Setter
                                {
                                    propiedad = Entrada .ScaleProperty,
                                    Valor = 1,5
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```

La única diferencia real entre el XAML y el código es el tratamiento de la Tipo de objetivo propiedad. En XAML, el Tipo de objetivo propiedad se establece en "Entrada" en cada uno de los tres Desencadenar definiciones. En el código, sin embargo, `typeof(Entrada)` se debe pasar como un argumento a la Desencadenar constructor. Si se comprueba la documentación de la Desencadenar clase, encontrará que la Tipo de objetivo la propiedad es de sólo conseguir. El analizador utiliza el XAML Tipo de objetivo Configuración de atributos para crear instancias de la Desencadenar objeto.

los Estilo clase también define una disparadores propiedad de tipo `IList<TriggerBase>`, lo que significa que se puede utilizar una Estilo para compartir Desencadenar objetos entre múltiples elementos. los `StyledTriggers` programa muestra cómo. Nótese que tanto el Estilo y el Desencadenar etiquetas contienen una Tipo de objetivo valor de la propiedad. los Estilo contiene una Setter objeto y usos `Style.Triggers` etiquetas de propiedad de elementos para el single Desencadenar objeto, que también contiene una Setter objeto:

```
< Pagina de contenido xmlns = "http://xamarin.com/schemas/2014/forms"
    xmlns: x = "http://schemas.microsoft.com/winfx/2009/xaml"
    x: Class = "StyledTriggers.StyledTriggersPage"
    Relleno = "20, 50, 120, 0" >

< ContentPage.Resources >
    < ResourceDictionary >
        < Estilo Tipo de objetivo = "Entrada" >
            < Setter Propiedad = "anchorX" Valor = "0" />
```

```

< Style.Triggers >
    < Desencadenar Tipo de objetivo = " Entrada " Propiedad = " Esta enfocado " Valor = " Ciento " >
        < Setter Propiedad = " Escala " Valor = " 1.5 " />
    </ Desencadenar >
</ Style.Triggers >
</ Estilo >
</ ResourceDictionary >
</ ContentPage.Resources >

< StackLayout Espaciado = " 20 " >

    < Entrada marcador de posición = " Ingrese su nombre " />

    < Entrada marcador de posición = " ingresa la dirección " />

    < Entrada marcador de posición = " entrar en la ciudad y el estado " />

</ StackLayout >
</ Página de contenido >

```

Porque el Estilo no tiene clave de diccionario, es un estilo implícito que se aplica automáticamente a todos los elementos de tipo Entrada. El individuo Entrada elementos sólo necesitan contener lo que es única para cada elemento.

Tal vez después de experimentar con la **entrypop** programa (o las dos variantes), decide que no desea que el Escala propiedad a simplemente "pop" a un valor de 1,5. ¿Quieres una animación. Usted quiere que se "hinche" de tamaño cuando gana el foco de entrada, y para ser animado de vuelta a la normalidad cuando pierde el foco de entrada.

Eso también es posible.

Desencadenar acciones y animaciones

Aunque algunos factores desencadenantes pueden ser realizados enteramente en XAML, otros requieren un poco de apoyo de código. Como ya saben, Xamarin.Forms no tiene apoyo directo para la implementación de animaciones en XAML, por lo que si desea utilizar un disparador para animar un elemento, se van a necesitar algo de código.

Hay un par de maneras de invocar una animación a partir de XAML. La manera más obvia es utilizar EventTrigger, que define dos propiedades:

- Evento de tipo cuerda.
- Comportamiento de tipo IList <TriggerAction>.

Cuando el elemento sobre el que se encuentra conectado el gatillo dispara ese evento en particular, la EventTrigger invoca toda la TriggerAction objetos en el Comportamiento colección.

Por ejemplo, VisualElement define dos eventos relacionados con el foco de entrada: centrado y Desenfocado. Puede configurar los nombres de eventos a la Evento propiedad de dos diferentes EventTrigger objetos. Cuando

el elemento desencadena el evento, los objetos de tipo TriggerAction se invocan. Su trabajo consiste en suministrar una clase que deriva de TriggerAction. Esta clase derivada anula un método denominado Invocar para responder al evento.

Xamarin.Forms define tanto una TriggerAction clase y una TriggerAction <T> clase, pero ambas clases son abstractas. Generalmente se le derivan de TriggerAction <T> y establecer el parámetro de tipo de la clase más generalizada la acción de disparo puede soportar.

Por ejemplo, supongamos que desea derivar de TriggerAction <T> para una clase que llama scaleTo para animar el Escala propiedad. Ajuste el parámetro de tipo de VisualElement porque esa es la clase que hace referencia el scaleTo método de extensión. Un objeto de este tipo también se pasa a Invocar.

Por convención, una clase que deriva de TriggerAction tiene una Acción sufijo en su nombre. una clase de este tipo puede ser tan simple como esto:

```
p\xedblico clase ScaleAction : TriggerAction < VisualElement >
{
    protegido override void Invocar( VisualElement visual )
    {
        visual.ScaleTo( 1.5 );
    }
}
```

Al incluir esta clase en una EventTrigger que est\xf3 unido a un Entrada ver, lo particular en-tratar objeto se pasa como argumento a la Invocar m\xedtodo, que anima que Entrada objetar el uso ScaleTo. Los Entrada se expande a 150 por ciento de su tama\xf1o original en una duraci\xf3n predeterminada de un cuarto de segundo.

Por supuesto, es probable que no quiere hacer la clase que especifica. as\xed de sencillo ScaleAction clase funcionar\xeda bien para el centrado evento, pero se necesitar\xfa una diferente para el Desenfocado evento para animar la Escala propiedad de nuevo por debajo de 1.

Tu Acci\xf3n <T> derivado puede incluir propiedades para hacer la clase muy generalizada. Puede incluso hacer que el ScaleAction la clase de modo generalizado que en esencia se convierte en un contenedor para el scaleTo m\xedtodo. Aqu\xed est\xfa la versi\xf3n de ScaleAction en el **Xamarin.FormsBook.Toolkit** biblioteca:

```
espacio de nombres Xamarin.FormsBook.Toolkit
{
    clase p\xiblica ScaleAction : TriggerAction < VisualElement >
    {
        p\xiblico ScaleAction()
        {
            // Configurar valores predeterminados.
            ancla = nuevo Punto( 0.5, 0.5 );
            Escala = 1;
            Longitud = 250;
            aliviar = aliviar.Lineal;
        }

        p\xiblico Punto ancla { conjunto; obtener; }
```

```

pública doble escala { conjunto ; obtener ; }

public int Longitud { conjunto ; obtener ; }

[ TypeConverter ( tipo de ( EasingConverter ))]
público aliviar aliviar { conjunto ; obtener ; }

protected void override Invocar( VisualElement visual )
{
    visual.AnchorX = Anchor.X;
    visual.AnchorY = Anchor.Y;
    visual.ScaleTo ( Escala, ( uint ) Longitud, aliviando );
}

}
}
}

```

Puede que se pregunte si debe hacer una copia de estas propiedades con las propiedades que pueden vincularse de manera que puedan ser blanco de enlaces de datos. No se puede hacer eso, sin embargo, porque TriggerAction deriva de Objeto más bien que BindableObject. Mantener las propiedades sencilla.

Observe la TypeConverter atribuir a la aliviar propiedad. Esta aliviar propiedad probablemente se establece en XAML, pero el analizador XAML no sabe cómo convertir cadenas de texto como "Springin" y "SinOut" a objetos de tipo Aliviar. El convertidor de tipos siguiendo la costumbre (también en **Xamarin.FormsBook.Toolkit**) asiste al analizador XAML en la conversión de cadenas de texto en aliviar objetos:

```

espacio de nombres Xamarin.FormsBook.Toolkit
{
    clase pública EasingConverter : TypeConverter
    {
        público bool anulación CanConvertFrom ( Tipo tipo de fuente )
        {
            Si (SourceType == nulo )
                arrojar nueva ArgumentNullException ( "EasingConverter.CanConvertFrom: sourceType" );

            regreso (SourceType == tipo de ( cuerda ));
        }

        público objeto de anulación ConvertFrom ( CultureInfo cultura, objeto valor )
        {
            Si (Valor == nulo || !(valor es una cadena ))
                return null ;

            cuerda name = (( cuerda ) Valor).trim ();

            Si (Name.StartsWith ( "Flexibilización" ))
            {
                name = name.Substring (7);
            }

            FieldInfo campo = tipo de ( aliviar ).GetRuntimeField (nombre);

            Si (Campo! = nulo && field.IsStatic)

```

```
        {
            regreso ( aliviar ) Field.GetValue ( nulo );
        }

        arrojar nueva InvalidOperationException (
            Cuerda .Formato( "No se puede convertir ` {0} ` en Xamarin.Forms.Easing" , Valor));
    }

}
```

los EntrySwell programa define un implícito Estilo para Entrada en su recursos diccionario. Ese Estilo tiene dos EventTrigger objetos en su disparadores colección, uno para centrado y el otro para Desenfocado. Tanto invocar una ScaleAction pero con diferentes valores de propiedades:

```
< Pagina de contenido xmlns = "http://xamarin.com/schemas/2014/forms" >
    xmlns: x = "http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns: kit de herramientas =
        "clr-espacio de nombres: Xamarin.FormsBook.Toolkit; montaje = Xamarin.FormsBook.Toolkit"
    x: Class = "EntrySwell.EntrySwellPage"
    Relleno = "20, 50, 120, 0" >

    < ContentPage.Resources >
        < ResourceDictionary >
            < Estilo Tipo de objetivo = " Entrada " >
                < Style.Triggers >
                    < EventTrigger Evento = " centrado " >
                        < kit de herramientas: ScaleAction Ancia = " 0, 0,5 " *
                            Escala = " 1.5 "
                            aliviar = " Brotará " />
                    </ EventTrigger >
                    < EventTrigger Evento = " Desenfocado " >
                        < kit de herramientas: ScaleAction Ancia = " 0, 0,5 " *
                            Escala = " 1 " />
                    </ EventTrigger >
                </ Style.Triggers >
            </ Estilo >
        </ ResourceDictionary >
    </ ContentPage.Resources >

    < StackLayout Espaciado = " 20 " >
        < Entrada marcador de posición = " Ingrese su nombre " />
        < Entrada marcador de posición = " ingresa la dirección " />
        < Entrada marcador de posición = " entrar en la ciudad y el estado " />
    </ StackLayout >
</ Pagina de contenido >
```

Darse cuenta de EventTrigger no requiere la Tipo de objetivo propiedad. El único constructor que EventTrigger define no tiene parámetros

Como cada Entrada recibe el foco de entrada, verá crecer grande y luego llega más allá brevemente el 1.5 Escala

valor. Ese es el efecto de la Brotará función de aceleración.

¿Qué pasa si desea utilizar una función de aceleración personalizada? Usted tendría que definir una función de este tipo de flexibilización en el código, por supuesto, y que puede hacer que en el archivo de código subyacente. Pero ¿cómo hacer referencia a que esa función de aceleración en XAML? Así es cómo:

En primer lugar, retire el ResourceDictionary etiquetas del archivo XAML. Esas etiquetas instancia del ResourceDictionary y ponerlo a la recursos propiedad.

En segundo lugar, en el constructor del archivo de código subyacente, una instancia del ResourceDictionary y ponerlo a la recursos propiedad. Haga esto antes InitializeComponent de modo que existe cuando se analiza el archivo XAML:

```
recursos = nuevo ResourceDictionary ();
InitializeComponent ();
```

En tercer lugar, entre esos dos estados, añadir una aliviar objeto con una función de aceleración personalizada a la recursos diccionario:

```
recursos = nuevo ResourceDictionary ();
Resources.Add ("CustomEase", nuevo aliviar (T => -6 * t * t + 7 * t));
InitializeComponent ();
```

Esta fórmula cuadrática mapas de 0 hasta 0 y 1 a 1, pero 0,5 a 2, por lo que será obvio si esta función de aceleración se utiliza correctamente por la animación.

Por último, hacer referencia a dicha entrada del diccionario utilizando StaticResource en el EventTrigger definición:

```
< EventTrigger Evento = " centrado " >
  < kit de herramientas: ScaleAction Anda = " 0, 0,5 "
    Escala = " 1,5 "
    aliviar = " {StaticResource CustomEase} " />
</ EventTrigger >
```

Debido a que el objeto en el recursos diccionario es de tipo facilitando, el analizador XAML asignará directamente a la aliviar propiedad de ScaleAction sin pasar por el TypeConverter.

Entre los ejemplos de código de este capítulo es una solución llamada **CustomEasingSwell** que muestra esta técnica.

No utilice DynamicResource para establecer la costumbre aliviar oponerse a la aliviar propiedad, tal vez con la esperanza de la definición de la función de aceleración de código en un momento posterior. DynamicResource requiere la propiedad de destino que estar respaldada por una propiedad enlazable; StaticResource no.

Usted ha visto cómo se puede utilizar Desencadenar establecer una propiedad en respuesta a un cambio de propiedad, y EventTrigger para invocar una TriggerAction objetar en respuesta a un evento de disparo.

Pero lo que si desea invocar una TriggerAction en respuesta a un cambio de propiedad? Tal vez desea invocar una animación a partir de XAML, pero no hay ningún evento apropiado para una EventTrigger.

Hay una segunda forma de invocar una TriggerAction derivado que implica la habitual Desencadenar

clase en lugar de EventTrigger. Si nos fijamos en la documentación de TriggerBase (la clase de la cual todas las otras clases de disparo derivan), verá las dos propiedades siguientes:

- EnterActions de tipo IList <TriggerAction>
- ExitActions de tipo IList <TriggerAction>

Cuando se utiliza con Desencadenar, todos TriggerAction objetos en el EnterActions colección se invoca cuando el Desencadenar condición sea verdadera, y todos los objetos de la ExitActions colección se invoca cuando la condición se convierte en falsa de nuevo.

los EnterExitSwell programa muestra esta técnica. Usa Desencadenar para supervisar el Esta enfocado propiedad e invoca dos instancias de ScaleAction para aumentar el tamaño de la Entrada cuando Esta enfocado se convierte Cíerto y para disminuir el tamaño de la Entrada cuando Esta enfocado deja de ser Cíerto:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    xmlns: kit de herramientas =
        "clr-espacio de nombres: Xamarin.FormsBook.Toolkit; montaje = Xamarin.FormsBook.Toolkit "
    x: Class = " EnterExitSwell.EnterExitSwellPage "
    Relleno = " 20, 50, 120, 0 " >

    < ContentPage.Resources >
        < ResourceDictionary >
            < Estilo Tipo de objetivo = " Entrada " >
                < Style.Triggers >
                    < Desencadenar Tipo de objetivo = " Entrada " Propiedad = " Esta enfocado " Valor = " Cíerto " >
                        < Trigger.EnterActions >
                            < kit de herramientas: ScaleAction Ancla = " 0, 0,5 "
                                Escala = " 1,5 "
                                aliviar = " Brotará " />
                        </ Trigger.EnterActions >

                        < Trigger.ExitActions >
                            < kit de herramientas: ScaleAction Ancla = " 0, 0,5 "
                                Escala = " 1 " />
                        </ Trigger.ExitActions >
                    </ Desencadenar >
                </ Style.Triggers >
            </ Estilo >
        </ ResourceDictionary >
    </ ContentPage.Resources >

    < StackLayout Espaciado = " 20 " >
        < Entrada marcador de posición = " Ingrese su nombre " />

        < Entrada marcador de posición = " ingresa la dirección " />

        < Entrada marcador de posición = " entrar en la ciudad y el estado " />
    </ StackLayout >
</ Página de contenido >
```

En resumen, puede invocar una clase derivada de `TriggerAction <T>` ya sea con un cambio en una propiedad mediante el uso de Desencadenar o con un evento de disparo mediante el uso de `EventTrigger`.

Pero no use EnterActions y ExitActions con EventTrigger. `EventTrigger` invoca sólo el `TriggerAction` objetos en su Comportamiento colección.

Más activadores de eventos

En el capítulo anterior en la animación mostró varios ejemplos de una Botón que hace girar o escalado a sí mismo cuando se ha hecho clic.

Mientras que la mayoría de esos ejemplos de animación fueron llevados a los extremos a los efectos de hacer demostraciones divertidas, no es razonable que una Botón para responder a un clic con un poco de animación. Este es un trabajo perfecto para `EventTrigger`.

Aquí está otro `TriggerAction` derivado. Es similar a `ScaleAction` pero incluye dos llamadas a `scaleTo` en lugar de sólo uno y por lo tanto se llama `ScaleUpAndDownAction`:

```
espacio de nombres Xamarin.FormsBook.Toolkit
{
    clase pública ScaleUpAndDownAction : TriggerAction < VisualElement >
    {
        público ScaleUpAndDownAction ()
        {
            ancla = nuevo Punto (0.5, 0.5);
            Escala = 2;
            Longitud = 500;
        }

        público Punto ancla { conjunto ; obtener ; }

        pública doble escala { conjunto ; obtener ; }

        public int Longitud { conjunto ; obtener ; }

        async protected void override Invocar(VisualElement visual)
        {
            visual.AnchorX = Anchor.X;
            visual.AnchorY = Anchor.Y;
            esperar visual.ScaleTo (Escala, ( uint ) Longitud / 2, aliviar .SinOut);
            esperar visual.ScaleTo (1, ( uint ) Longitud / 2, aliviar .SinIn);
        }
    }
}
```

Esta clase duros códigos de la aliviar funciones para mantener el código simple.

los `ButtonGrowth` programa define un intrínseca Estilo que establece tres Botón propiedades y incluye una `EventTrigger` que invoca `ScaleUpAndDownAction` con parámetros por defecto en respuesta a la hecho clic evento:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
      xmlns:x = " http://schemas.microsoft.com/winfx/2009/xaml "
```

```

xmlns: kit de herramientas =
    "clr-espacio de nombres: Xamarin.Forms.Book.Toolkit; montaje = Xamarin.Forms.Book.Toolkit "
x: Class = "ButtonGrowth.ButtonGrowthPage"

< ContentPage.Resources >
    < ResourceDictionary >
        < Estilo Tipo de objetivo = " Botón " >
            < Setter Propiedad = " HorizontalOptions " Valor = " Centrar " />
            < Setter Propiedad = " VerticalOptions " Valor = " CenterAndExpand " />
            < Setter Propiedad = " Tamaño de fuente " Valor = " Grande " />

        < Style.Triggers >
            < EventTrigger Evento = " hecho clic " >
                < kit de herramientas: ScaleUpAndDownAction />
            </ EventTrigger >
        </ Style.Triggers >
    </ Estilo >
</ ResourceDictionary >
</ ContentPage.Resources >

< StackLayout >
    < Botón Texto = " Botón 1 " />
    < Botón Texto = " Botón # 2 " />
    < Botón Texto = " Botón # 3 " />
</ StackLayout >
</ Página de contenido >

```

Aquí hay tres botones, ya que han crecido en tamaño en respuesta a clics:



¿Habría sido posible utilizar dos instancias de ScaleAction aquí en lugar de ScaleUpAnd-DownAction ejemplo -ona que escaló la Botón en el tamaño y el otro que escalan hacia abajo? No.

Sólo estamos tratando con un evento de la hecho clic eventos y todo tiene que ser invocado cuando se dispara el evento. Un EventTrigger sin duda puede invocar múltiples acciones, pero estas acciones ocurren simultáneamente. Dos ScaleAction instancias que se ejecutan simultáneamente serían luchar entre sí.

Sin embargo, no es una solución. Aquí está un DelayedScaleAction clase que deriva de ScaleAction pero incluye una Task.Delay llamar antes de la scaleTo llamada:

```
espacio de nombres Xamarin.FormsBook.Toolkit
{
    clase pública DelayedScaleAction : ScaleAction
    {
        público DelayedScaleAction (): base ()
        {

            // Configurar valores predeterminados.
            Delay = 0;
        }

        public int plazo de tiempo { conjunto ; obtener ; }

        asincrono protected void override Invocar(VisualElement visual)
        {
            visual.AnchorX = Anchor.X;
            visual.AnchorY = Anchor.Y;
            esperar Tarea .Delay (Delay);
            esperar visual.ScaleTo (Escala, ( uint ) Longitud, aliviando);
        }
    }
}
```

Ahora puede modificar el **ButtonGrowth XAML** archivo para incluir dos DelayedScaleAction objetos provocada por la hecho clic evento. Estos son a la vez invocado simultáneamente, pero la segunda tiene su Retrasar propiedad establecida en el mismo valor que la Longitud propiedad de la primera, por lo que la primera scaleTo termina como el segundo scaleTo comienza:

```
<Estilo Tipo de objetivo = " Botón " >
...
< Style.Triggers >
    < EventTrigger Evento = " hecho clic " >
        < kit de herramientas: DelayedScaleAction Escala = " 2 "
            Longitud = " 250 "
            aliviar = " SinOut " />

        < kit de herramientas: DelayedScaleAction Retrasar = " 250 "
            Escala = " 1 "
            Longitud = " 250 "
            aliviar = " SinIn " />
    </ EventTrigger >
</ Style.Triggers >
</ Estilo >
```

DelayedScaleAction es un poco más difícil de usar que ScaleUpAndDownAction, pero es más

flexibles, y también se puede definir clases nombradas DelayedTranslateAction y DelayedRotateAction. Acción añadir a la mezcla.

En el capítulo anterior se vio una Botón derivado llamado JiggleButton que se ejecuta una breve animación cuando el Botón se hace clic. Se trata de un tipo de animación que se puede implementar utilizando alternativamente una TriggerAction. La ventaja es que se puede usar con la normal Botón clase, y potencialmente separar el efecto de un tipo particular de vista y un evento en particular por lo que podría ser utilizado con otros puntos de vista y otros eventos.

Aquí está un TriggerAction derivado que implementa el mismo tipo de animación como JiggleButton pero con tres propiedades para que sea más flexible. Para distinguir más claramente del código anterior, el nombre de esta clase es ShiverAction:

```
espacio de nombres Xamarin.FormsBook.Toolkit
{
    clase pública ShiverAction : TriggerAction < VisualElement >
    {
        público ShiverAction ()
        {
            Longitud = 1000;
            Ángulo = 15;
            Vibraciones = 10;
        }

        public int Longitud { conjunto ; obtener ; }

        pública doble angle { conjunto ; obtener ; }

        public int vibraciones { conjunto ; obtener ; }

        protegido override void Invocar( VisualElement visual )
        {
            visual.Rotation = 0;
            visual.AnchorX = 0,5;
            visual.AnchorY = 0,5;
            visual.RotateTo( Angle, ( uint )Longitud,
                nuevo aliviar ( T => Mates .Pecado( Mates t .PI * *
                    Mates .Pecado( Mates .PI * 2 * Vibraciones * t)));
            }
        }
    }
}
```

Darse cuenta de Invocar inicializa el Rotación propiedad del elemento visual objetivo a cero. Esto es para evitar problemas cuando las Botón se pulsa dos veces en sucesión y Invocar se llama, mientras que la animación anterior todavía se está ejecutando.

El archivo XAML de la ShiverButtonDemo programa define un implícito Estilo que incluye la ShiverAction con los valores más extremos fijados a sus tres propiedades:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    xmlns: kit de herramientas =
```

```

    "clr-espacio de nombres: Xamarin.FormsBook.Toolkit; montaje = Xamarin.FormsBook.Toolkit "
x:Class = "ShiverButtonDemo.ShiverButtonDemoPage"

< ContentPage.Resources >
    < ResourceDictionary >
        < Estilo Tipo de objetivo = "Botón" >
            < Setter Propiedad = "HorizontalOptions" Valor = "Centrar" />
            < Setter Propiedad = "VerticalOptions" Valor = "CenterAndExpand" />
            < Setter Propiedad = "Tamaño de fuente" Valor = "Grande" />

            < Style.Triggers >
                < EventTrigger Evento = "hecho clic" >
                    < kit de herramientas: ShiverAction Longitud = "3000" 
                        Ángulo = "45" 
                        vibraciones = "25" />
                </ EventTrigger >
            </ Style.Triggers >
        </ Estilo >
    </ ResourceDictionary >
</ ContentPage.Resources >

< StackLayout >
    < Botón Texto = "Botón 1" />
    < Botón Texto = "Botón #2" />
    < Botón Texto = "Botón #3" />
</ StackLayout >
</ Pagina de contenido >

```

El tres Botón elementos comparten la misma instancia de ShiverAction, pero cada llamada a la En-
voke método es para una específica Botón objeto. temblor de cada botón es independiente de los demás.

Pero lo que si desea utilizar ShiverAction responder a aprovechado eventos en un elemento en lugar de
hecho clic eventos en un Botón -por ejemplo, para hacer vibrar una Marco con un cierto contenido, o un *Imagen*? los
aprovechado evento sólo se define por TapGestureRecognizer, pero no se puede adjuntar una EventTrigger a una TapGestureRecognizer porque
TapGestureRecognizer no tiene una disparadores colección. Tampoco se puede adjuntar una EventTrigger a una Ver objeto y especificar
el aprovechado evento. Ese aprovechado
evento no se puede encontrar en la Ver objeto.

La solución es utilizar un comportamiento, como se demostrará más adelante en este capítulo.

También es posible utilizar EventTrigger objetos para la validación de entrada. Aquí está un TriggerAction derivado llamado NumericValidationAction
con un argumento genérico de Entrada, por lo que sólo se aplica a en-
trar puntos de vista. Cuando Invocar se llama, el argumento es una Entrada objeto, por lo que puede acceder a las propiedades específicas de Entrada, en
este caso Texto y Color de texto. El método comprueba si el Texto propiedad de la en-
trar se puede analizar en un válido doble. Si no es así, el texto es de color rojo para alertar al usuario:

```

espacio de nombres Xamarin.FormsBook.Toolkit
{
    clase pública NumericValidationAction : TriggerAction < Entrada >
    {
        protected void override Invocar( Entrada entrada )
        {

```

```
        doble resultado;
        bool isValid = Doble.TryParse(entry.Text, fuera resultado);
        entry.TextColor = isValid? Color.Defecto : Color.Rojo;
    }
}
}
```

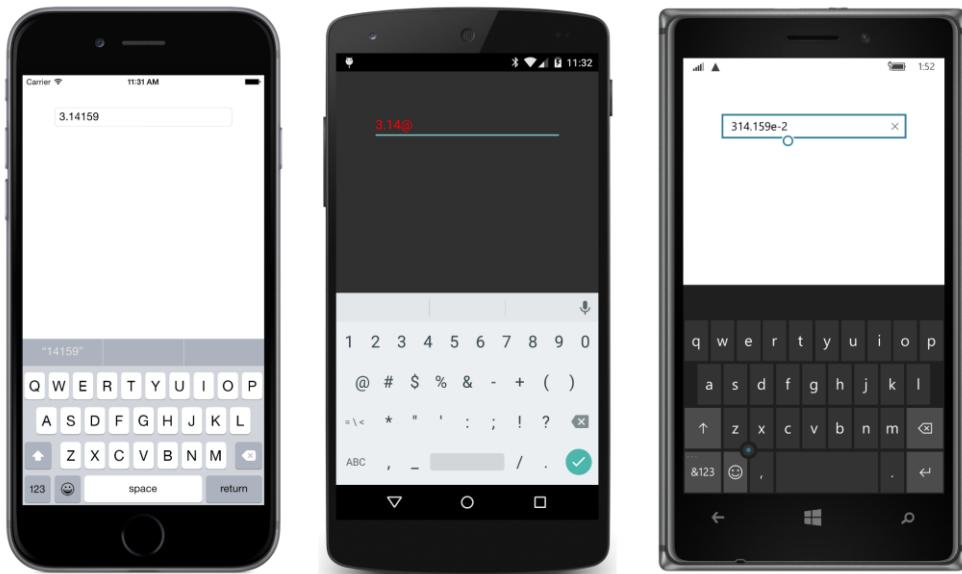
Puede adjuntar el código a una Entrada con un EventTrigger Para el TextChanged evento, como se demuestra en el **TriggerEntryValidation** programa:

```
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    xmlns: kit de herramientas =
        "clr-espacio de nombres: Xamarin.FormsBook.Toolkit; montaje = Xamarin.FormsBook.Toolkit"
    x: Class = " TriggerEntryValidation.TriggerEntryValidationPage "
    Relleno = " 50 " >

    < StackLayout >
        < Entrada marcador de posición = " Introduzca un System.Double " >
            < Entry.Triggers >
                < EventTrigger Evento = " TextChanged " >
                    < kit de herramientas: NumericValidationAction />
                </ EventTrigger >
            </ Entry.Triggers >
        </ Entrada >
    </ StackLayout >
</ Pagina de contenido >
```

Siempre que los cambios en el texto, la Invocar método de NumericValidationAction se llama.

La captura de pantalla muestra las entradas numéricas válidas para los dispositivos móviles iOS y Windows 10, pero un número no válido en el dispositivo Android:



Por desgracia, esto no funciona del todo bien en la plataforma Windows universal: Si un número no válido se escribe en el Entrada, el texto se vuelve rojo cuando el Entrada pierde el foco de entrada. Sin embargo, funciona muy bien en las otras plataformas de Windows en tiempo de ejecución (Windows 8.1 y Windows Phone 8.1).

disparadores de datos

Hasta el momento, sólo se ha visto desencadenantes que operan dentro del contexto de un objeto en particular. UN Desencadenar responde a un cambio en una propiedad de un objeto cambiando otra propiedad de ese mismo objeto, o mediante la invocación de una Acción que afecta a ese objeto. Los EventTrigger responden de manera similar a un evento disparado por un objeto para invocar una Acción en ese mismo objeto.

Los DataTrigger es diferente. Como el otro TriggerBase derivados, la DataTrigger está unido a un elemento visual o definido en una Estilo. Sin embargo, el DataTrigger puede detectar un cambio en la propiedad *otro* objeto a través de un enlace de datos, y, o bien cambiar una propiedad en el objeto que está unido a, o (mediante el uso de la EnterActions y ExitActions colección heredado de Desencadenar-Base) invocar una TriggerAction en ese objeto.

DataTrigger define las tres propiedades siguientes.

- Unión de tipo BindingBase.
- Valor de tipo Objeto.
- setters de tipo IList<Setter>. Esta es la propiedad del contenido DataTrigger.

Desde la perspectiva de un programa de aplicación, la DataTrigger es muy similar a Desencadenar excepto que la propiedad de Desencadenar llamado Propiedad se sustituye con el Unión propiedad. Ambos Desencadenar

y DataTrigger requerir la Tipo de objetivo la propiedad que desea ajustar.

¿Cuál es el otro objeto que el Unión propiedad de DataTrigger referencia? Puede ser parte de un modelo de vista en un escenario de MVVM u otro elemento de la página.

Se puede recordar la **SchoolOfFineArt** biblioteca del Capítulo 19, "La Colección de vistas." Estudiante clase en la biblioteca que define una propiedad denominada Sexo de tipo cuerda que se ha configurado como "masculino" o "femenino". Los **GenderColors** programa que se presenta a continuación utiliza esa propiedad en conjunto con una DataTrigger establecer un color azul o rosado para el estudiante (sin tener en cuenta cómo irremediablemente anticuado podría parecer que el esquema de color).

UN Vista de la lista muestra todos los estudiantes de la escuela secundaria, y una ViewCell formatea cada estudiante para mostrar la foto del estudiante, nombre completo, y el promedio de calificaciones actual:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    xmlns: la escuela = "clr-espacio de nombres: SchoolOfFineArt; montaje = SchoolOfFineArt "
    x: Class = "GenderColors.GenderColorsPage" >

< ContentPage.Padding >
    < OnPlatform x: TypeArguments = "Espesor" >
        iOS = "0, 20, 0, 0" />
    </ ContentPage.Padding >

    < ContentPage.BindingContext >
        < la escuela: SchoolViewModel />
    </ ContentPage.BindingContext >

    < StackLayout BindingContext = "0 La unión StudentBody" >
        < Etiqueta Texto = "(Escuela) Encuadernación" >
            Tamaño de fuente = "Grande"
            FontAttributes = "Negrita"
            HorizontalTextAlignment = "Centrar" />

        < Vista de la lista ItemsSource = "Los estudiantes {Binding}" >
            VerticalOptions = "FillAndExpand" >
                < ListView.RowHeight >
                    < OnPlatform x: TypeArguments = "x: Int32" >
                        iOS = "70"
                        Androide = "70"
                        WinPhone = "100" />
                </ ListView.RowHeight >

                < ListView.ItemTemplate >
                    < DataTemplate >
                        < ViewCell >
                            < Cuadrícula Relleno = "0, 5" >
                                < Grid.ColumnDefinitions >
                                    < ColumnDefinition Anchura = "80" />
                                    < ColumnDefinition Anchura = "*" />
                                </ Grid.ColumnDefinitions >

                                < Imagen Grid.Column = "0" >
                                    Fuente = "0 La unión PhotoFilename"
                                </ Imagen >
                            </ ViewCell >
                        </ DataTemplate >
                    </ ListView.ItemTemplate >
                < /ListView >
            < /Vista de la lista >
        < /Etiqueta >
    < /StackLayout >
</ Página de contenido >
```

```

VerticalOptions = "Centrar" />

<StackLayout Grid.Column = "1" 
             VerticalOptions = "Centrar" >
<Etiqueta Texto = "[] La unión NombreCompleto" 
           Tamaño de fuente = "22" 
           Color de texto = "Rosado" >
<Label.Triggers >
  <DataTrigger Tipo de objetivo = "Etiqueta" 
                Unión = "[] La unión sexual" 
                Valor = "Masculino" >
    <Setter Propiedad = "Color de texto" Valor = "# 8080FF" />
  </DataTrigger >
</Label.Triggers >
</Etiqueta >

<Etiqueta Texto = "{Binding GradePointAverage, 
               StringFormat = 'GPA = {0:F2}'}" 
           Tamaño de fuente = "discursiva" />
</StackLayout >
</Cuadricula >
</ViewCell >
</DataTemplate >
</ListView.ItemTemplate >
</Vista de la lista >
</StackLayout >
</Pagina de contenido >

```

Los usos de los programas ViewCell más bien que ImageCell, por lo que tiene acceso a una Etiqueta sobre la que se puede adjuntar una DataTrigger. Un disparador no se puede conectar directamente a una Celda o Celda derivado porque no hay disparadores colección definida para estas clases.

los Etiqueta muestra el Nombre completo propiedad de la Estudiante objeto y la Color de texto se establece en Rosado. Sin embargo, una DataTrigger comprueba si el Sexo propiedad de la Estudiante objeto es igual "masculino", y si es así que utiliza una Setter para establecer el Color de texto a un color azul claro. Aquí es que Etiqueta aislado del resto de la célula:

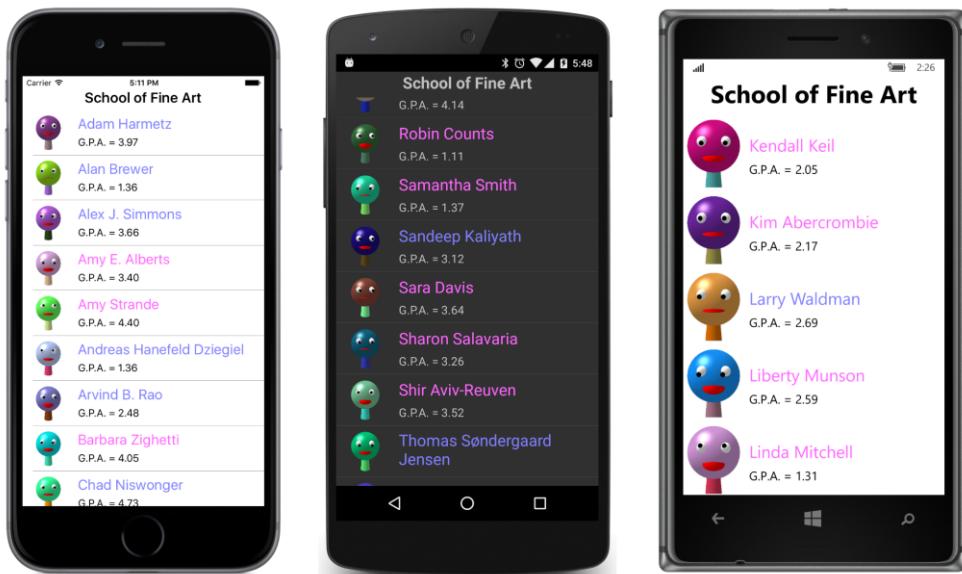
```

<Etiqueta Texto = "[] La unión NombreCompleto" 
           Tamaño de fuente = "Grande" 
           Color de texto = "Rosado" >
<Label.Triggers >
  <DataTrigger Tipo de objetivo = "Etiqueta" 
                Unión = "[] La unión sexual" 
                Valor = "Masculino" >
    <Setter Propiedad = "Color de texto" Valor = "# 8080FF" />
  </DataTrigger >
</Label.Triggers >
</Etiqueta >

```

los BindingContext del DataTrigger es la misma que la BindingContext del Etiqueta a la que está unido. Ese BindingContext es un particular, Estudiante objeto, por lo que la Unión sobre el DataTrigger sólo tiene que especificar el Sexo propiedad.

Aquí está en acción:



Algo similar se puede hacer con un enlace de datos directamente desde el **Sexo** propiedad de la **Estudiante** oponerse a la **Color** de texto propiedad de la etiqueta (**o el ImageCell**), pero requeriría un convertidor de unión. Los **DataTrigger** hace el trabajo sin ningún código adicional.

Sin embargo, por sí mismo la DataTrigger no se puede imitar la ColorCodedStudents programa en el Capítulo

19. Ese programa muestra un estudiante en rojo si el promedio de calificaciones del estudiante cae peligrosamente por debajo de un criterio 2.0. La comparación numérica menos-que requiere un poco de código. Esto también es un trabajo para un comportamiento y una vez que aprender acerca de los comportamientos más adelante en este capítulo, usted debería ser capaz de codificar algo como esto por sí mismo.

También es posible que DataTrigger para hacer referencia a otro elemento de la página para vigilar una propiedad de ese elemento.

Por ejemplo, una de las tareas clásicas en entornos gráficos es deshabilitar un botón si nada se ha escrito en un campo de entrada de texto.

Tal vez el campo de entrada de texto es un nombre de archivo y el botón ejecuta un código para cargar o guardar el archivo. No tiene ningún sentido para que el botón se active si el nombre del archivo está en blanco.

Puede hacer ese trabajo en su totalidad en XAML con una DataTrigger. Aquí está el margen de beneficio en el ButtonEnabler proyecto:

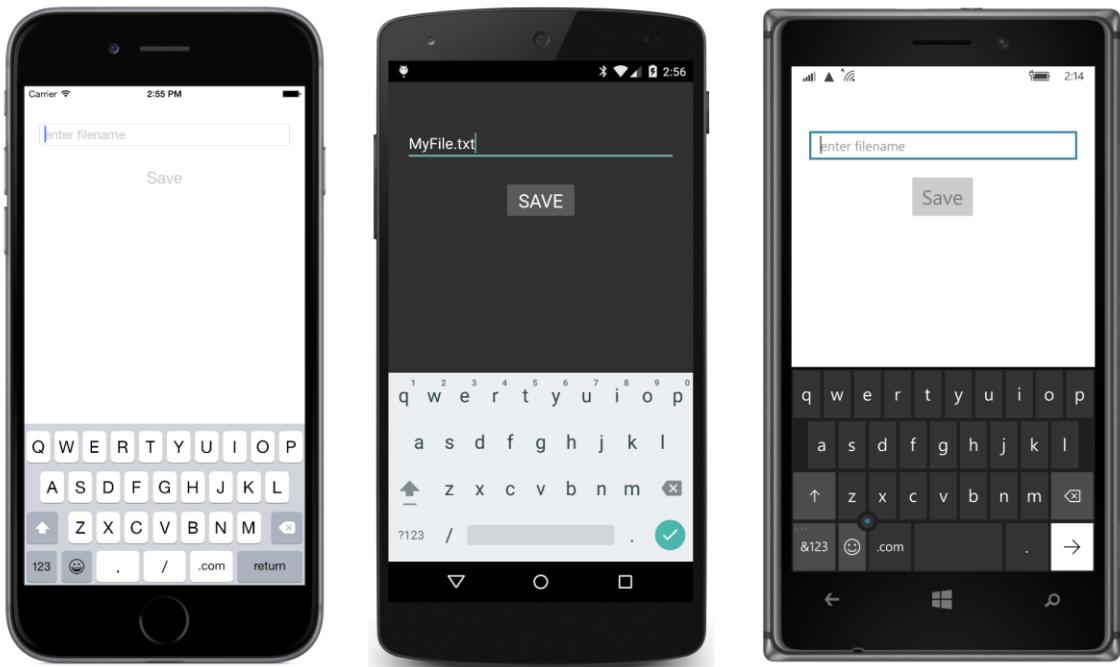
```
<Pagina de contenido xmlns = "http://xamarin.com/schemas/2014/forms"
    xmlns : X = "Http://schemas.microsoft.com/winfx/2009/xaml"
    X : Clase = "ButtonEnabler.ButtonEnablerPage"
    Relleno = "20, 50">
```

```
< StackLayout Espaciado = "20">
    < Entrada X : Nombre = "Entrada"
        Texto = ""
        Teclado = "URL"
        marcador de posición = "Introducir el nombre de archivo" />

    < Botón Texto = "Guardar"
        Tamaño de fuente = "Large"
        HorizontalOptions = "Center">
        < Button.Triggers >
            < DataTrigger Tipo de objetivo = "Botón"
                Unión = "{ Unión Fuente = { X : Referencia entrada },
                    Camino = Text.length}"
                Valor = "0">
                < Setter Propiedad = "isEnabled" Valor = "False" />
            </ DataTrigger >
        </ Button.Triggers >
    </ Botón >
</ StackLayout >
</ Página de contenido >
```

los DataTrigger sobre el Botón fija su Unión propiedad con una Fuente haciendo referencia a la Entrada elemento. los Camino se establece en Text.length. los Texto propiedad de la Entrada elemento es de tipo cuerda, y Longitud es una propiedad de cuerda, por lo que esta unión se refiere al número de caracteres introducidos en el Entrada elemento. los Valor propiedad de DataTrigger se ajusta a cero, por lo que cuando hay cero caracteres introducidos en el Entrada, el Setter se invoca la propiedad, que establece el Está habilitado propiedad de la Botón a Falso.

Sobre la base de la entrada en el Entrada elemento, el Botón está deshabilitado en el iPhone y Windows Mobile 10 pantallas que se muestran aquí, pero habilitadas en la pantalla de Android:



Aunque esto representa una pequeña mejora en la interfaz de usuario, si no tiene una DataTrigger que había necesidad de implementar esta mejora en el código de una TextChanged manejador de la Entrada, o lo que se necesita para escribir un convertidor de unión para una Unión Entre los Está habilitado propiedad de la Botón y el text.length propiedad de la Entrada.

El archivo XAML ButtonEnabler contiene un valor de la propiedad fundamental que no puede haber notado:

<Entrada ... Texto = " " ... />

Cuando un Entrada se crea en primer lugar, la Texto la propiedad no es una cadena vacía, pero nulo, lo que significa que el enlace de datos en el DataTrigger está tratando de hacer referencia a la Longitud propiedad de una nulo objeto de cadena, y se producirá un error. Debido a que falla la unión, la Botón se activará cuando el programa se inicia por primera vez. Sólo se desactiva después de que el usuario escribe un carácter y retrocesos.

la inicialización de la Texto propiedad a una cadena vacía no tiene otro efecto, sino para permitir que el DataTrigger a trabajar cuando el programa se pone en marcha.

La combinación de las condiciones en el MultiTrigger

Ambos Desencadenar y el DataTrigger un seguimiento eficaz de una propiedad para determinar si es igual a un valor determinado. Eso se llama el gatillo de condición, y si la condición es verdadera, entonces una colección de Setter objetos se invocan.

Como programador, es posible que comience a preguntarse si se pueden tener varias condiciones en un disparador. Pero una vez que se empieza a hablar de varias condiciones, es necesario determinar si desea combinar condiciones con una operación lógica OR o AND operación si el disparador se invoca si *alguna* las condiciones son verdaderas, o si se requiere que *todas* las condiciones son verdaderas.

Si quieres un disparador invoca cuando se cumple todo-lo lógico y múltiples condiciones de casos y que es la última de las cuatro clases que se derivan de `TriggerBase`. Los `MultiTrigger` define dos propiedades de la colección:

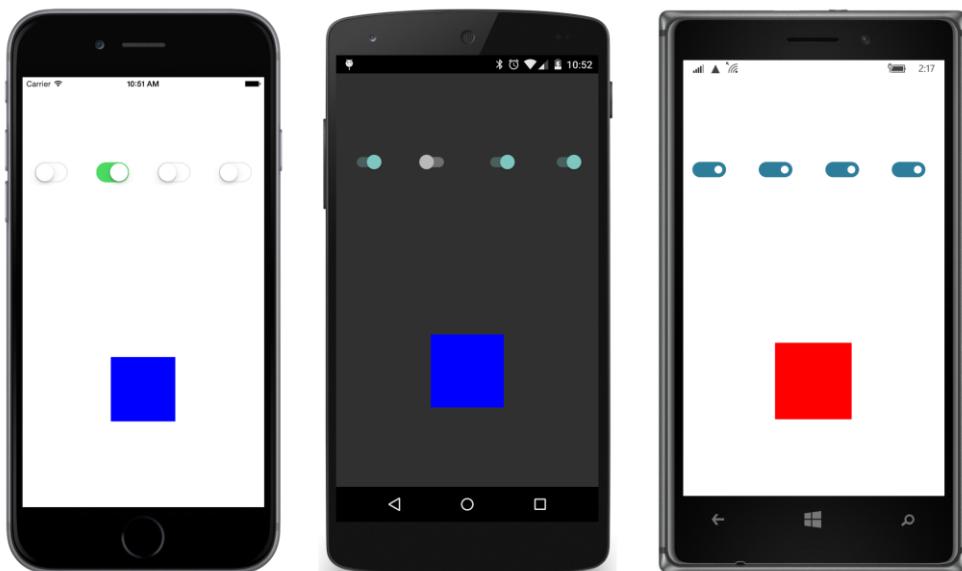
- condiciones de tipo `IList <condición>`
- setters de tipo `IList <Setter>`

Condición es una clase abstracta y tiene dos clases de descendientes:

- `PropertyCondition`, que tiene Propiedad y Valor propiedades como Desencadenar
- `BindingCondition`, que tiene Unión y Valor propiedades como `DataTrigger`

Se pueden mezclar múltiples `PropertyCondition` y `BindingCondition` objetos en el mismo condición colección de la `MultiTrigger`. Cuando todas las condiciones son verdaderas, toda la `Setter` objetos en el setters se aplican colección.

Veamos un ejemplo sencillo: En el `Y condiciones` programa, de cuatro Cambiar elementos de compartir la página con un azul `BoxView`. Cuando todo el Cambiar elementos están activados, la `BoxView` se vuelve rojo:



El archivo XAML muestra cómo se hace esto. Los disparadores colección de la `BoxView` contiene una Multi-

Desencadenar. los Tipo de objetivo Se requiere la propiedad. los condiciones colección contiene cuatro Unión-Condición objetos, cada uno de los cuales hace referencia a la IsToggled propiedad de uno de los cuatro Cambiar elementos y los controles para una Cierta valor. Si todas las condiciones son verdaderas, la MultiTrigger establece el Color propiedad de la BoxView a Rojo:

```
< Pagina de contenido xmlns = "http://xamarin.com/schemas/2014/forms"
    xmlns : X = "Http://schemas.microsoft.com/winfx/2009/xaml"
    X : Clase = "AndConditions.AndConditionsPage">

< StackLayout >
    < Cuadrícula VerticalOptions = "CenterAndExpand" >
        < Cambiar X : Nombre = "Switch1" Grid.Column = "0"
            HorizontalOptions = "Center" />

        < Cambiar X : Nombre = "Switch2" Grid.Column = "1"
            HorizontalOptions = "Center" />

        < Cambiar X : Nombre = "Switch3" Grid.Column = "2"
            HorizontalOptions = "Center" />

        < Cambiar X : Nombre = "Switch4" Grid.Column = "3"
            HorizontalOptions = "Center" />
    </ Cuadrícula >

    < BoxView WidthRequest = "100"
        HeightRequest = "100"
        VerticalOptions = "CenterAndExpand"
        HorizontalOptions = "Centro"
        Color = "Blue" >
        < BoxView.Triggers >
            < MultiTrigger Tipo de objetivo = "BoxView" >
                < MultiTrigger.Conditions >
                    < BindingCondition Unión = "{ Unión Fuente = { X : Referencia switch1 },
                        Camino = IsToggled}" 
                        Valor = "True" />

                    < BindingCondition Unión = "{ Unión Fuente = { X : Referencia switch2 },
                        Camino = IsToggled}" 
                        Valor = "True" />

                    < BindingCondition Unión = "{ Unión Fuente = { X : Referencia switch3 },
                        Camino = IsToggled}" 
                        Valor = "True" />

                    < BindingCondition Unión = "{ Unión Fuente = { X : Referencia Switch4 },
                        Camino = IsToggled}" 
                        Valor = "True" />
                </ MultiTrigger.Conditions >
                < Setter Propiedad = "Color" Valor = "Red" />
            </ MultiTrigger >
        </ BoxView.Triggers >
    </ BoxView >
</ StackLayout >
</ Pagina de contenido >
```

Y esa es la combinación. ¿Qué pasa con una combinación O?

Porque el disparadores colección puede acomodar múltiples DataTrigger objetos, se podría pensar que esto funcionaría:

```
<BoxView WidthRequest = "100"
         HeightRequest = "100"
         VerticalOptions = "CenterAndExpand"
         HorizontalOptions = "Centro"
         Color = "Blue">

<BoxView.Triggers >
    <DataTrigger Tipo de objetivo = "BoxView"
                 Unión = "{ Unión Fuente = { X : Referencia switch1 }, Camino = IsToggled}"
                 Valor = "True">
        <Setter Propiedad = "Color" Valor = "Red" />
    </DataTrigger >

    <DataTrigger Tipo de objetivo = "BoxView"
                 Unión = "{ Unión Fuente = { X : Referencia switch2 }, Camino = IsToggled}"
                 Valor = "True">
        <Setter Propiedad = "Color" Valor = "Red" />
    </DataTrigger >

    <DataTrigger Tipo de objetivo = "BoxView"
                 Unión = "{ Unión Fuente = { X : Referencia switch3 }, Camino = IsToggled}"
                 Valor = "True">
        <Setter Propiedad = "Color" Valor = "Red" />
    </DataTrigger >

    <DataTrigger Tipo de objetivo = "BoxView"
                 Unión = "{ Unión Fuente = { X : Referencia Switch4 }, Camino = IsToggled}"
                 Valor = "True">
        <Setter Propiedad = "Color" Valor = "Red" />
    </DataTrigger >
</BoxView.Triggers >
</BoxView >
```

Y si lo intenta, puede encontrarse con que sí parece funcionar al principio. Pero a medida que experimentar más con convertir varios Cambiar elementos de encendido y apagado, usted encontrará que lo que realmente hace *no* trabajo.

Si debe o no debe trabajar trabajar está abierto a debate. El cuatro DataTrigger todos los objetos de destinatarios con el mismo Color propiedad, y si cada DataTrigger trabaja de forma independiente para determinar si esa Setter se debe aplicar o no, entonces esto realmente no debería funcionar como un operador lógico OR.

Sin embargo, tenga en cuenta las leyes de matemático Augustus De Morgan de estilo victoriano de la lógica, que el estado (usando sintaxis de C # para AND, OR, negación lógica, y la equivalencia):

$$\bullet | == \bullet ! (\text{!} \bullet \& \bullet)$$

$$\bullet \text{ y } \bullet == ! (\bullet | ! \bullet)$$

Esto significa que puede utilizar MultiTrigger para realizar una operación lógica O como el **OrConditions** programa demuestra:

```

< Pagina de contenido xmlns = "http://xamarin.com/schemas/2014/forms"
    xmlns : X = "Http://schemas.microsoft.com/winfx/2009/xaml"
    X : Clase = "OrConditions.OrConditionsPage">

< StackLayout >
    < Cuadrícula VerticalOptions = "CenterAndExpand">
        < Cambiar X : Nombre = "Switch1" Grid.Column = "0"
            HorizontalOptions = "Center" />

        < Cambiar X : Nombre = "Switch2" Grid.Column = "1"
            HorizontalOptions = "Center" />

        < Cambiar X : Nombre = "Switch3" Grid.Column = "2"
            HorizontalOptions = "Center" />

        < Cambiar X : Nombre = "Switch4" Grid.Column = "3"
            HorizontalOptions = "Center" />
    </ Cuadrícula >

    < BoxView WidthRequest = "100"
        HeightRequest = "100"
        VerticalOptions = "CenterAndExpand"
        HorizontalOptions = "Centro"
        Color = "Red">
        < BoxView.Triggers >
            < MultiTrigger Tipo de objetivo = "BoxView">
                < MultiTrigger.Conditions >
                    < BindingCondition Unión = "{ Unión Fuente = { X : Referencia switch1 },
                        Camino = IsToggled}">
                        Valor = "False" />

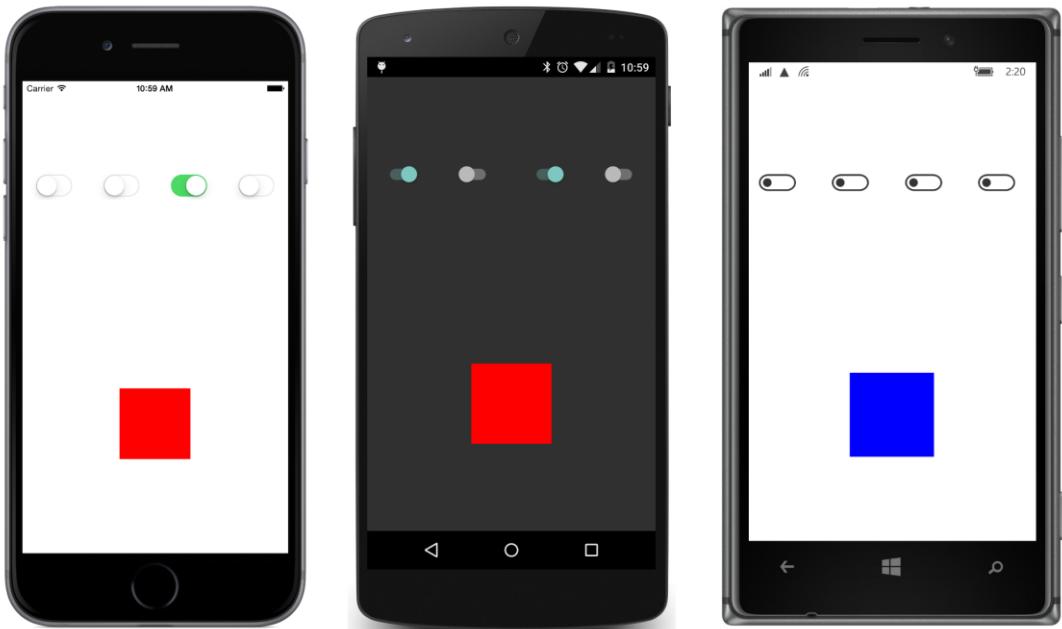
                    < BindingCondition Unión = "{ Unión Fuente = { X : Referencia switch2 },
                        Camino = IsToggled}">
                        Valor = "False" />

                    < BindingCondition Unión = "{ Unión Fuente = { X : Referencia switch3 },
                        Camino = IsToggled}">
                        Valor = "False" />

                    < BindingCondition Unión = "{ Unión Fuente = { X : Referencia Switch4 },
                        Camino = IsToggled}">
                        Valor = "False" />
                </ MultiTrigger.Conditions >
                < Setter Propiedad = "Color" Valor = "Blue" />
            </ MultiTrigger >
        </ BoxView.Triggers >
    </ BoxView >
</ StackLayout >
</ Pagina de contenido >

```

Es lo mismo que **Y condiciones** excepto toda la lógica se da la vuelta alrededor. Todos BindingCondition objetos de verificación para una Falso valor de la IsToggled propiedad, y si se cumplen todas las condiciones, el normalmente roja BoxView de color azul:



Aquí hay otra manera de pensar en estos dos programas: En **Y condiciones**, el BoxView siempre es azul a menos que todos los Cambiar elementos estén activados. En **OrConditions**, el BoxView es siempre de color rojo a menos que todos los Cambiar elementos estén apagados.

Supongamos que tenemos un escenario en el que dos Entrada campos y una Botón. Desea habilitar la Botón si alguno Entrada campo contiene algún texto.

Voltar la lógica al revés: ¿Realmente desea desactivar el Botón si ambos Entrada campos no contienen texto. Eso es bastante fácil:

```
< StackLayout >
  < Entrada X : Nombre = "Entry1"
    Texto = "" />

  < Entrada X : Nombre = "Entrada2"
    Texto = "" />

  < Botón Texto = "Enviar">
    < Button.Triggers >
      < MultiTrigger Tipo de objetivo = "Button">
        < MultiTrigger.Conditions >
          < BindingCondition Unión = "{ Unión Fuente = { X : Referencia entry1 },
            Camino = Text.length}">
            Valor = "0" />
          < BindingCondition Unión = "{ Unión Fuente = { X : Referencia entrada2 },
            Camino = Text.length}">
            Valor = "0" />
        </ MultiTrigger.Conditions >
      </ MultiTrigger >
    </ Button.Triggers >
  </ Botón >
```

```

    < Setter Propiedad = "IsEnabled" Valor = "False" />
    </ MultiTrigger >
    </ Button.Triggers >
    </ Botón >
</ StackLayout >

```

Tenga en cuenta que los dos Entrada inicializar los campos Texto propiedad a una cadena vacía para que la propiedad no es igual a nulo. Si ambos Texto propiedades tienen una longitud de cero, entonces los dos BindingConditions están satisfechos y el Está habilitado propiedad de la Botón se establece en Falso.

Sin embargo, no es tan fácil de adaptar esta opción para activar el Botón sólo si *ambos* Entrada puntos de vista tienen algún texto. Si se intenta volcar la lógica alrededor, debe cambiar el BindingCondition objetos para que comprueben la existencia de una Texto propiedad con una longitud *no* igual a cero, y eso no es una opción.

Para ayudar a hacer realidad la lógica, puede utilizar algún intermediario invisible Cambiar elementos:

```

< StackLayout >
    < Entrada X : Nombre = "Entry1"
        Texto = "" />

    < Cambiar X : Nombre = "Switch1"
        Es visible = "False">
        < Switch.Triggers >
            < DataTrigger Tipo de objetivo = "Switch"
                Unión = "{ Unión Fuente = { X : Referencia entry1 },
                    Camino = Text.length}"
                Valor = "0">
                < Setter Propiedad = "IsToggled" Valor = "True" />
            </ DataTrigger >
        </ Switch.Triggers >
    </ Cambiar >

    < Entrada X : Nombre = "Entrada2"
        Texto = "" />

    < Cambiar X : Nombre = "Switch2"
        Es visible = "False">
        < Switch.Triggers >
            < DataTrigger Tipo de objetivo = "Switch"
                Unión = "{ Unión Fuente = { X : Referencia entrada2 },
                    Camino = Text.length}"
                Valor = "0">
                < Setter Propiedad = "IsToggled" Valor = "True" />
            </ DataTrigger >
        </ Switch.Triggers >
    </ Cambiar >

    < Botón Texto = "Enviar"
        Está habilitado = "False">
        < Button.Triggers >
            < MultiTrigger Tipo de objetivo = "Button">
                < MultiTrigger.Conditions >

```

```

< BindingCondition Unión = "{ Unión Fuente = { X : Referencia switch1 },
    Camino = IsToggled}">
    Valor = "False" />
< BindingCondition Unión = "{ Unión Fuente = { X : Referencia switch2 },
    Camino = IsToggled}">
    Valor = "False" />
</ MultiTrigger.Conditions >

< Setter Propiedad = "isEnabled" Valor = "True" />
</ MultiTrigger>
</ Botón >
</ StackLayout >

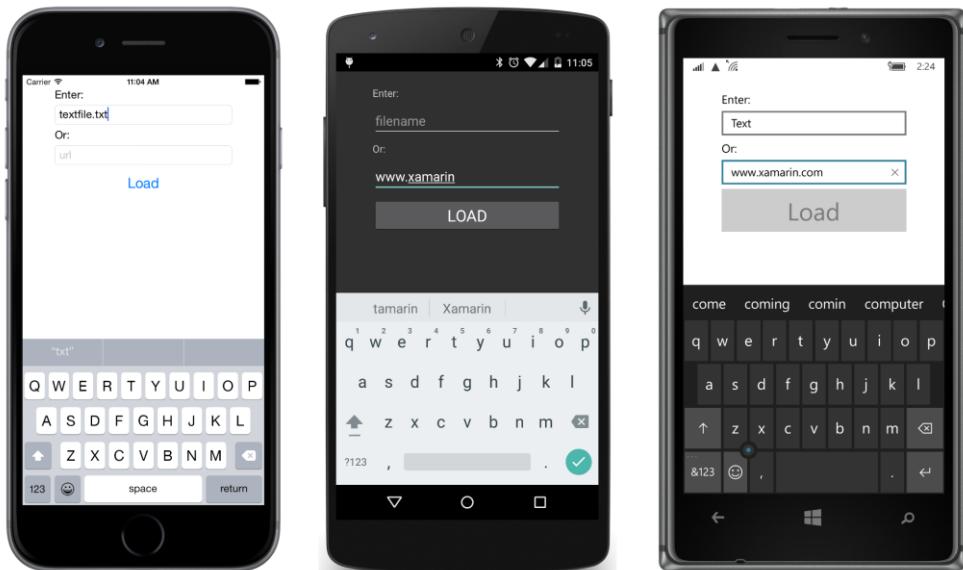
```

Cada Entrada ahora tiene un compañero Cambiar que utiliza un DataTrigger para establecer su IsToggled propiedad a Cierto si la longitud de la Texto propiedad de la Entrada es cero. Los dos Cambiar elementos se pueden utilizar en el MultiTrigger. Si ambos Cambiar elementos tienen su IsToggled propiedades ajustado a Cierto,

entonces ambos Entrada campos contienen algo de texto y la Está habilitado propiedad de la Botón se puede configurar para Cierto.

Si desea combinar realidad AND y OR operaciones, tendrá que participar en algunos niveles más profundos de la lógica.

Por ejemplo, suponga que tiene un escenario con dos Entrada vistas y una Botón, y el Botón Debe estar activado solamente si alguno de los dos Entrada vistas contiene texto, pero no tanto si Entrada vistas contienen un texto:



Tal vez (como se sugiere captura de pantalla) una de las Entrada puntos de vista es un nombre de archivo y la otra es para una dirección URL, y el programa tiene una y sólo una de estas dos cadenas de texto.

Lo que necesita es una operación OR exclusiva (XOR), y es una combinación de AND, OR, y los operadores de negación:

$$\bullet \wedge == \bullet (\bullet | \bullet) \text{ y! } (\bullet \text{ y } \bullet)$$

Esto se puede hacer con tres MultiTrigger objetos, dos de los cuales están en intermedio invisible

Cambiar elementos y el último está en la Botón sí mismo. Aquí está la **XorConditions** archivo XAML con comentarios que describen las diferentes piezas de la lógica:

```
< Pagina de contenido xmlns = "http://xamarin.com/schemas/2014/forms"
    xmlns : X = "Http://schemas.microsoft.com/winfx/2009/xaml"
    X : Clase = "XorConditions.XorConditionsPage"
    Relleno = "50, 20">

    < StackLayout >
        < Etiqueta Texto = "Enter:" />

        < Entrada X : Nombre = "Entry1"
            Texto = ""
            marcador de posición = "Nombre de archivo" />

        <! - IsToggled es cierto si entry1 no tiene ningún texto ->
        < Cambiar X : Nombre = "Switch1"
            Es visible = "False">
            < Switch.Triggers >
                < DataTrigger Tipo de objetivo = "Switch"
                    Unión = "{ Unión Fuente = { X : Referencia entry1 },
                    Camino = Text.length}"
                    Valor = "0">
                    < Setter Propiedad = "IsToggled" Valor = "True" />
                </ DataTrigger >
            </ Switch.Triggers >
        </ Cambiar >

        < Etiqueta Texto = "O:" />

        < Entrada X : Nombre = "Entrada2"
            Texto = ""
            marcador de posición = "Url" />

        <! - IsToggled es cierto si entrada2 no tiene ningún texto ->
        < Cambiar X : Nombre = "Switch2"
            Es visible = "False">
            < Switch.Triggers >
                < DataTrigger Tipo de objetivo = "Switch"
                    Unión = "{ Unión Fuente = { X : Referencia entrada2 },
                    Camino = Text.length}"
                    Valor = "0">
                    < Setter Propiedad = "IsToggled" Valor = "True" />
                </ DataTrigger >
            </ Switch.Triggers >
        </ Cambiar >

        <! - IsToggled es verdadero si la entrada tiene un poco de texto (operación O) ->
```

```

< Cambiar X : Nombre = "Switch3"
  IsToggled = "True"
  Es visible = "False">
  < Switch.Triggers >
    < MultiTrigger Tipo de objetivo = "Switch">
      < MultiTrigger.Conditions >
        < BindingCondition Unión = "{ Unión Fuente = { X : Referencia switch1 },
          Camino = IsToggled}">
          Valor = "True" />
        < BindingCondition Unión = "{ Unión Fuente = { X : Referencia switch2 },
          Camino = IsToggled}">
          Valor = "True" />
      </ MultiTrigger.Conditions >

      < Setter Propiedad = "IsToggled" Valor = "False" />
    </ MultiTrigger >
  </ Switch.Triggers >
</ Cambiar >

<! - IsToggled es cierto si tanto de entrada tiene algo de texto (operación Y) ->
< Cambiar X : Nombre = "Switch4"
  Es visible = "False">
  < Switch.Triggers >
    < MultiTrigger Tipo de objetivo = "Switch">
      < MultiTrigger.Conditions >
        < BindingCondition Unión = "{ Unión Fuente = { X : Referencia switch1 },
          Camino = IsToggled}">
          Valor = "False" />
        < BindingCondition Unión = "{ Unión Fuente = { X : Referencia switch2 },
          Camino = IsToggled}">
          Valor = "False" />
      </ MultiTrigger.Conditions >

      < Setter Propiedad = "IsToggled" Valor = "True" />
    </ MultiTrigger >
  </ Switch.Triggers >
</ Cambiar >

<! - botón se activa si cualquiera de entrada tiene algo de texto, pero no ambos (operación XOR) ->
< Botón Texto = "Load"
  Está habilitado = "False"
  Tamaño de fuente = "Large">
  < Button.Triggers >
    < MultiTrigger Tipo de objetivo = "Button">
      < MultiTrigger.Conditions >
        < BindingCondition Unión = "{ Unión Fuente = { X : Referencia switch3 },
          Camino = IsToggled}">
          Valor = "True" />
        < BindingCondition Unión = "{ Unión Fuente = { X : Referencia Switch4 },
          Camino = IsToggled}">
          Valor = "False" />
      </ MultiTrigger.Conditions >

      < Setter Propiedad = "IsEnabled" Valor = "True" />
    </ MultiTrigger >
  </ Button.Triggers >

```

```

</ MultiTrigger >
</ Button.Triggers >
</ Botón >
</ StackLayout >
</ Pagina de contenido >

```

Por supuesto, una vez que el XAML obtiene esta extravagante, nadie culpa que si simplemente decide activar o desactivar el Botón ¡en código!

comportamientos

Disparadores y comportamientos se discuten generalmente en tandem, ya que tienen cierta superposición de la aplicación industrial. A veces podrás desconcertado si se debe utilizar un disparador o el comportamiento, ya sea porque parece hacer el trabajo.

Cualquier cosa que puede hacer con un disparador también se puede ver con un comportamiento. Sin embargo, un comportamiento siempre implica un cierto código, que es una clase que deriva de Comportamiento <T>. Se dispara sólo implicará código si está escribiendo un Acción <T> derivado de una EventTrigger o por EnterActions o ExitActions colecciones de los otros factores desencadenantes.

Obviamente, si se puede hacer lo que es necesario el uso de uno de los detonantes sin necesidad de escribir código, entonces no utilice un comportamiento. Pero a veces no es tan clara.

Vamos a comparar un disparador y un comportamiento que haga el mismo trabajo.

los TriggerEntryValidation programa mostrado anteriormente en este capítulo se utiliza una clase llamada NumericEntryAction que comprueba si un número escrito en una Entrada vista califica como un válido doble valor y los colores rojos del texto si no es así:

```

espacio de nombres Xamarin.Forms.Toolkit
{
    clase pública NumericValidationAction : TriggerAction < Entrada >
    {
        protegido override void Invocar( Entrada entrada )
        {
            doble resultado;
            bool isValid = Doble .TryParse (entry.Text, fuera resultado);
            entry.TextColor = isValid? Color .Defecto : Color .Rojo;
        }
    }
}

```

Esto se hace referencia en una EventTrigger unido a una Entrada:

```

< Entrada marcador de posición = "Introduzca un System.Double">
    < Entry.Triggers >
        < EventTrigger Evento = "TextChanged">
            < kit de herramientas : NumericValidationAction />
        </ EventTrigger >
    </ Entry.Triggers >

```

</ Entrada >

Se puede utilizar un comportamiento de este mismo trabajo. El primer paso es derivar una clase de Comportamiento <T>. El argumento genérico es la clase base más generalizada de que el comportamiento puede manejar. En este ejemplo, que es una Entrada ver. A continuación, anular dos métodos virtuales, llamados OnAttachedTo y OnDetaching-

De los OnAttachedTo método se llama cuando el comportamiento se adjunta a un objeto visual en particular, y le da a su comportamiento oportunidad de inicializarse. A menudo esto implica unir algunos controladores de eventos al objeto. Los OnDetachingFrom método se llama cuando el comportamiento se retira del objeto visual. Incluso si esto ocurre sólo cuando el programa está terminando, se debe deshacer cualquier cosa que el OnAttachedTo método hace.

Aquí está la NumericValidationBehavior clase:

```
espacio de nombres Xamarin.FormsBook.Toolkit
{
    clase pública NumericValidationBehavior : Comportamiento < Entrada >
    {
        protected void override OnAttachedTo ( Entrada entrada )
        {
            base .OnAttachedTo ( entrada );
            entry.TextChanged += OnEntryTextChanged;
        }

        protected void override OnDetachingFrom ( Entrada entrada )
        {
            base .OnDetachingFrom ( entrada );
            entry.TextChanged -= OnEntryTextChanged;
        }

        vacío OnEntryTextChanged ( objeto remitente, TextChangedEventArgs args )
        {
            doble resultado;
            bool isValid = Doble .TryParse ( args.NewTextValue, fuera resultado );
            (( Entrada ) Remitente) .TextColor = isValid? Color .Defecto : Color .Rojo;
        }
    }
}
```

Los OnAttachedTo método concede un controlador para el TextChanged caso de la Entrada, y el OnDetachingFrom método desprende que manejador. El manejador sí lo hace el mismo trabajo que el Invocar método en el NumericValidationAction.

Porque el NumericValidationBehavior clase instala el controlador para el TextChanged evento, el comportamiento puede ser utilizado sin especificar nada más allá del nombre de la clase. Aquí está el archivo XAML para el BehaviorEntryValidation programa, que se diferencia del programa anterior que utiliza una EventTrigger especificando el comportamiento en un estilo implícito que se aplica a las cuatro Entrada puntos de vista:

```
< Página de contenido xmlns = "http://xamarin.com/schemas/2014/forms"
      xmlns : X = "Http://schemas.microsoft.com/winfx/2009/xaml"
      xmlns : kit de herramientas =
          "Clr-espacio de nombres: Xamarin.FormsBook.Toolkit; montaje = Xamarin.FormsBook.Toolkit"
```

```

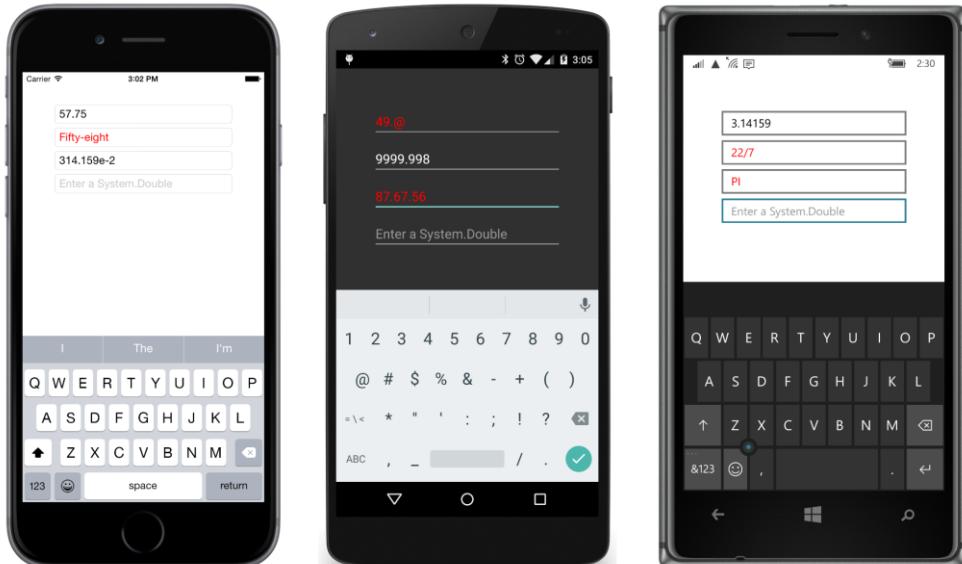
X : Clase = "BehaviorEntryValidation.BehaviorEntryValidationPage"
Relleno = "50"

< ContentPage.Resources >
    < ResourceDictionary >
        < Estilo Tipo de objetivo = "Entrada" >
            < Style.Behaviors >
                < kit de herramientas : NumericValidationBehavior />
            </ Style.Behaviors >
        </ Estilo >
    </ ResourceDictionary >
</ ContentPage.Resources >

< StackLayout >
    < Entrada marcador de posición = "Introduzca un System.Double" />
    < Entrada marcador de posición = "Introduzca un System.Double" />
    < Entrada marcador de posición = "Introduzca un System.Double" />
    < Entrada marcador de posición = "Introduzca un System.Double" />
</ StackLayout >
</ Página de contenido >

```

Esta Estilo objeto es compartida entre los cuatro Entrada puntos de vista, por lo que sólo un único NumericValidationBehavior objeto se crea una instancia. Como este único objeto se adjunta a cada uno de los cuatro Entrada puntos de vista, que concede una TextChanged manejador en cada uno para que el único NumericValidationBehavior objeto opera de forma independiente en los cuatro puntos de vista:



En este ejemplo particular, el TriggerAction sería preferido sobre el Comportamiento porque es

menos código y el código no se refiere a un evento en particular, por lo que es más generalizada.

Sin embargo, un comportamiento puede ser tan generalizada o tan específico como desee, y los comportamientos también tienen la capacidad de participar más plenamente en el archivo XAML a través de enlaces de datos.

Comportamientos con propiedades

los Comportamiento <T> clase se deriva de la Comportamiento clase, que deriva de BindableObject. Esto sugiere que su Comportamiento <T> derivado puede definir sus propias propiedades enlazables.

Más temprano que viste alguna Acción <T> derivados tales como ScaleAction y ShiverAction que definen algunas propiedades para darles más flexibilidad. Sin embargo, una Comportamiento <T> derivado puede definir propiedades enlazables que pueden servir como propiedades de origen para enlaces de datos. Esto significa que usted no tiene que codificar el comportamiento de modificar una propiedad particular, como el establecimiento de la Color de texto propiedad de una

Entrada a Rojo. En su lugar puede decidir más tarde cómo desea que el comportamiento afecta a la interfaz de usuario, y poner en práctica ese derecho en el archivo XAML. Esto le da al comportamiento una mayor cantidad de flexibilidad y permite que el XAML para jugar un papel más importante en el aspecto de la conducta que se refiere a la interfaz de usuario.

Aquí está una clase en el **Xamarin.FormsBook.Toolkit** biblioteca llamada ValidEmailBehavior, que es similar a NumericValidationBehavior excepto que utiliza una expresión regular para determinar si el Texto propiedad de una Entrada es una dirección válida de correo electrónico:

```
espacio de nombres Xamarin.FormsBook.Toolkit
{
    clase pública ValidEmailBehavior : Comportamiento < Entrada >
    {
        estático solo lectura BindablePropertyKey IsValidPropertyKey =
            BindableProperty .CreateReadOnly ( "Es válida" ,
                tipo de ( bool ),
                tipo de ( ValidEmailBehavior ),
                falso );

        sólo lectura estática pública BindableProperty IsValidProperty =
            IsValidPropertyKey.BindableProperty;

        public bool Es válida
        {
            conjunto privado ( EstablecerValor ( IsValidPropertyKey, valor ); }
            obtener { regreso ( bool ) GetValue ( IsValidProperty ); }
        }

        protected void override OnAttachedTo ( Entrada entrada )
        {
            entry.TextChanged + = OnEntryTextChanged;
            base .OnAttachedTo ( entrada );
        }

        protected void override OnDetachingFrom ( Entrada entrada )
        {
            entry.TextChanged - = OnEntryTextChanged;
            base .OnDetachingFrom ( entrada );
        }
    }
}
```

```

        }

        vacio OnEntryTextChanged ( objeto remitente, TextChangedEventArgs args)
        {
            Entrada entrada = ( Entrada )remitente;
            IsValid = IsValidEmail ( entry.Text);
        }

        bool IsValidEmail ( cuerda FijEntrada)
        {
            Si ( Cuerda .IsNullOrEmpty ( FijEntrada))
                falso retorno ;

            tratar
            {
                // desde https://msdn.microsoft.com/en-us/library/01escwtf(v=vs.110).aspx
                regreso expresiones regulares .IsMatch (FijEntrada,
                    @".^((?:"")|"+(<|\\).??!" "@)|.?!. (((0-9a-zA-Z)((\\))|^+
                    @|-#\\$%&`^*+|=\\^\\{\\}|!?-\\W))|^+
                    @"(?<=[0-9a-zA-Z])@)|)?(\\)([0D{1,3}\\{3}\\d{1,3}\\{1}]|^+
                    @"(([0-9a-zA-Z]-\\w)*[0-9a-zA-Z]*|[a-zA-Z0-9].[0,22][a-zA-Z0-9]))$",
                    RegexOptions .Ignorar caso, Espacio de tiempo .FromMilliseconds (250));
            }
            captura ( RegexMatchTimeoutException )
            {
                falso retorno ;
            }
        }
    }
}

```

En lugar de establecer la Texto propiedad de la Entrada a Rojo, ValidEmailBehavior define una Es válida
la propiedad que está respaldado por una propiedad enlazable. Debido a que no tiene sentido para el código externo a esta clase para establecer el Es válida propiedad, es una propiedad enlazable de sólo lectura. los Bindable.CreateRead-
Solamente llamada crea una clave enlazable la propiedad privada que se utiliza por el Valor ajustado llamar en el sector privado conjunto
descriptor de acceso de Es válida. El público IsValidProperty propiedad que puede vincularse está referenciado por el GetValue
llamar como de costumbre.

La forma de utilizar que Es válida propiedad es totalmente de usted.

Por ejemplo, el **EmailValidationDemo** programa se une que Es válida alojamiento hasta el Es visible
propiedad de una Imagen mostrar una imagen "pulgador hacia arriba". Que "pulgador hacia arriba" mapa de bits se sienta encima de otro
Imagen elemento con un "pulgador hacia abajo" para indicar cuando una dirección válida de correo electrónico se ha escrito. Ese Es-
Válido establecimiento también está ligado a la Está habilitado propiedad de una Enviar botón. Observe que el Fuente de los dos enlaces de datos
es el ValidEmailBehavior objeto:

```

< Pagina de contenido xmlns = "http://xamarin.com/schemas/2014/forms"
    xmlns : X = "Http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns : local = "CLR-espacio de nombres: EmailValidationDemo"
    xmlns : kit de herramientas =
        "Cir-espacio de nombres: Xamarin.FormsBook.Toolkit; montaje = Xamarin.FormsBook.Toolkit"
    X : Clase = "EmailValidationDemo.EmailValidationDemoPage"

```

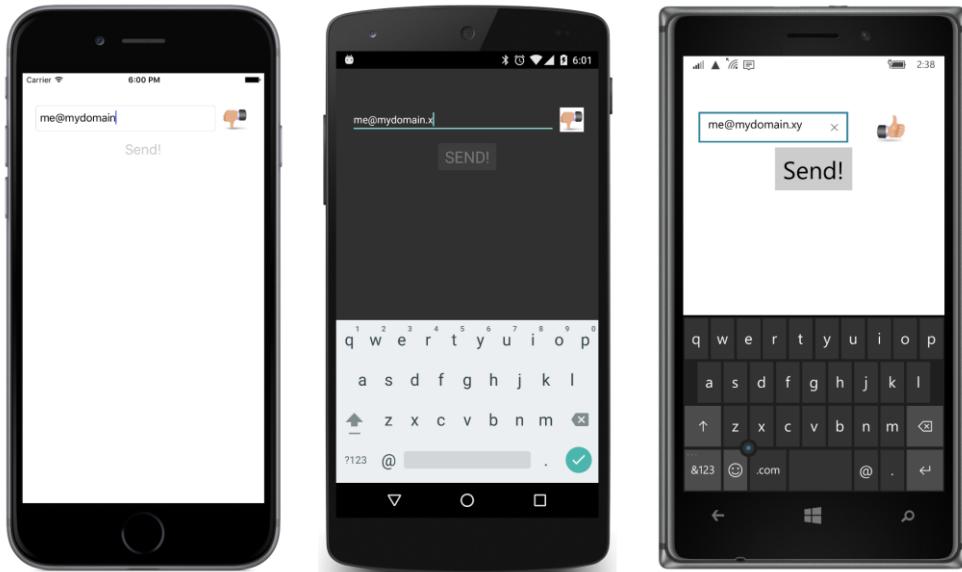
```
Relleno = "20, 50">

< StackLayout >
    < StackLayout Orientación = "Horizontal" >
        < Entrada marcador de posición = "Introduzca la dirección de correo electrónico" >
            Teclado = "Enviar"
            HorizontalOptions = "FillAndExpand" >
                < Entry.Behaviors >
                    < kit de herramientas : ValidEmailBehavior X : Nombre = "ValidEmail" />
                </ Entry.Behaviors >
            </ Entrada >

        < Cuadrícula HeightRequest = "40" >
            < Imagen Fuente =
                "{ local : ImageResource EmailValidationDemo .Images.ThumbsDown.png }" >
                < Imagen Fuente = "{ local : ImageResource EmailValidationDemo .Images.ThumbsUp.png }" >
                    Es visible = "{ Unión Fuente = { X : Referencia email valido },
                        Camino = IsValid }" />
                </ Cuadrícula >
            </ StackLayout >

            < Botón Texto = "Enviar" >
                Tamaño de fuente = "Large"
                HorizontalOptions = "Centro"
                Está habilitado = "{ Unión Fuente = { X : Referencia email valido },
                    Camino = IsValid }" />
            </ StackLayout >
        </ Pagina de contenido >
```

Como usted está escribiendo una dirección de correo electrónico, no se considera válida hasta que tenga al menos un dominio de nivel superior de dos caracteres:



Los dos mapas de bits son parte de la común `EmailValidationDemo` proyecto. los `ImageResource` clase extensión de marcado utilizado para hacer referencia a los mapas de bits se discutió en el capítulo 13, "mapas de bits", y que debe ser parte del mismo conjunto que contiene los mapas de bits:

```
espacio de nombres EmailValidationDemo
{
    [ ContentProperty ("Fuente" )]
    clase pública ImageResourceExtension : IMarkupExtension
    {
        public string Fuente { obtener ; conjunto ; }

        objeto público ProvideValue ( IServiceProvider proveedor de servicio)
        {
            Si (Fuente == nulo )
            return null ;

            regreso Fuente de imagen .FromResource (Fuente);
        }
    }
}
```

¿Qué pasa si usted tiene múltiples Entrada vistas en la misma página que hay que comprobar si las direcciones de correo electrónico válidas.
¿Podría incluir la `ValidEmailBehavior` clase en una comportamientos cobro de una `Estilo`?

No, no puedes. los `ValidEmailBehavior` clase define una propiedad denominada Es válida. Esto significa que un caso particular de `ValidEmailBehavior` siempre tiene un estado particular, que es el valor de esta propiedad. Esto tiene una implicación significativa:

Un comportamiento que mantiene el estado como un campo o una propiedad no puede ser compartida, lo que significa que no debe incluirse en una Estilo.

Si es necesario utilizar `ValidEmailBehavior` para múltiples Entrada vistas en la misma página, no lo ponen en una Estilo. Añadir una instancia independiente de la comportamientos colecciones de cada uno de los Entrada puntos de vista.

La ventaja de este Es válida propiedad supera las desventajas, sin embargo, porque se puede utilizar la propiedad en una variedad de maneras. Aquí hay un programa llamado `EmailValidationConverter` que utiliza el Es válida propiedad con un convertidor de unión ya en el `Xamarin.FormsBook.Toolkit` biblioteca para elegir entre dos cadenas de texto:

```
< Página de contenido xmlns = "http://xamarin.com/schemas/2014/forms"
    xmlns : X = "Http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns : kit de herramientas =
        "Clí-espacio de nombres: Xamarin.FormsBook.Toolkit; montaje = Xamarin.FormsBook.Toolkit"
    X : Clase = "EmailValidationConverter.EmailValidationConverterPage"
    Relleno = "50">

    < StackLayout >
        < StackLayout Orientación = "Horizontal" >
            < Entrada marcador de posición = "Introduzca la dirección de correo electrónico" >
                HorizontalOptions = "FillAndExpand" >
                    < Entry_Behaviors >
                        < kit de herramientas : ValidEmailBehavior X : Nombre = "ValidEmail" />
                    </ Entry_Behaviors >
                </ Entrada >

                < Etiqueta HorizontalTextAlignment = "Centro" >
                    VerticalTextAlignment = "Center" >
                    < label.text >
                        < Unión Fuente = "{ X : Referencia_email valido }" >
                            Camino = "IsValid" >
                            < Binding.Converter >
                                < kit de herramientas : BoolToObjectConverter X : TypeArguments = "X: String" >
                                    FalseObject = "Todavía no!" >
                                    TrueObject = "OK!" />
                            </ Binding.Converter >
                        </ Unión >
                    </ label.text >
                </ Etiqueta >
            </ StackLayout >

            < Botón Texto = "Enviar" >
                Tamaño de fuente = "Large" >
                HorizontalOptions = "Centro" >
                Está habilitado = "{ Unión Fuente = { X : Referencia_email valido },
                    Camino = IsValid}"/>
            </ Botón >
        </ StackLayout >
    </ Página de contenido >
```

los `BoolToObjectConverter` escoge entre las dos cadenas de texto "Todavía no!" y "OK!".

Sin embargo, usted puede hacer esto mismo con un poco de lógica más sencilla y sin convertidor vinculante mediante el uso de una `DataTrigger`, como el `EmailValidationTrigger` programa demuestra. El "1 Todavía no ha sido" el texto se asigna a la `Texto` propiedad de la `Etiqueta`, mientras que una `DataTrigger` sobre el `Etiqueta` contiene una

unión a la Es válida propiedades para establecer el texto "OK!":

```
< Pagina de contenido xmlns = "http://xamarin.com/schemas/2014/forms"
    xmlns : X = "Http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns : kit de herramientas =
        "Clr-espacio de nombres: Xamarin.FormsBook.Toolkit; montaje = Xamarin.FormsBook.Toolkit"
    X : Clase = "EmailValidationTrigger.EmailValidationTriggerPage"
    Relleno = "50">

    < StackLayout >
        < StackLayout Orientación = "Horizontal" >
            < Entrada marcador de posición = "Introduzca la dirección de correo electrónico" >
                HorizontalOptions = "FillAndExpand" >
                    < Entry.Behaviors >
                        < kit de herramientas : ValidEmailBehavior X : Nombre = "ValidEmail" />
                    </ Entry.Behaviors >
                </ Entrada >

                < Etiqueta Texto = "Todavía no!" >
                    HorizontalTextAlignment = "Centro"
                    VerticalTextAlignment = "Center" >
                    < Label.Triggers >
                        < DataTrigger Tipo de objetivo = "Etiqueta" >
                            Unión = "{ Unión Fuente = { X : Referencia email valido },
                                Camino = IsValid}" >
                            Valor = "True" >
                            < Setter Propiedad = "Texto" Valor = "OK!" />
                        </ DataTrigger >
                    </ Label.Triggers >
                </ Etiqueta >
            </ StackLayout >

            < Botón Texto = "Enviar" >
                Tamaño de fuente = "Large"
                HorizontalOptions = "Centro" >
                Está habilitado = "{ Unión Fuente = { X : Referencia email valido },
                    Camino = IsValid}"/>
            </ StackLayout >
        </ Pagina de contenido >
```

Hacer referencia a un comportamiento de una unión en un conjunto de datos DataTrigger es una técnica poderosa.

Altera y casillas de verificación

En el capítulo 15, "La interfaz interactiva", y el Capítulo 16, "El enlace de datos", que vio cómo construir tradicional Caja puntos de vista. Sin embargo, otro enfoque de vistas personalizadas es incorporar la lógica interactiva de las vistas en sus comportamientos y luego darse cuenta de los efectos visuales en su totalidad en XAML. Este enfoque le da la flexibilidad de personalizar los elementos visuales con marcadores en vez de código. Debido a que el aspecto visual no es parte de la lógica subyacente, puede crear efectos visuales especiales cada vez que se utiliza el comportamiento.

Aquí está una clase en el **Xamarin.FormsBook.Toolkit** biblioteca llamada **ToggleBehavior**. Al igual que los **Xamarin.Forms Cambiar** elemento, se define una propiedad denominada **IsToggled** que está respaldado por una enlazable

propiedad. **ToggleBehavior** simplemente instala una **TapGestureRecognizer** a lo visual que está al lado de y cambia el estado de la **IsToggled** la propiedad siempre que se detecte un grifo:

```
espacio de nombres Xamarin.FormsBook.Toolkit
{
    clase pública ToggleBehavior : Comportamiento < Ver >
    {
        TapGestureRecognizer tapRecognizer;

        sólo lectura estática pública BindableProperty IsToggledProperty =
            BindableProperty .Create < ToggleBehavior , bool > ( Tb => tb.IsToggled, falso );

        public bool IsToggled
        {
            conjunto { EstablecerValor ( IsToggledProperty, valor ); }
            obtener { regreso ( bool ) GetValue ( IsToggledProperty ); }
        }

        protected void override OnAttachedTo ( Ver ver )
        {
            base .OnAttachedTo ( vista );

            tapRecognizer = nuevo TapGestureRecognizer ();
            tapRecognizer.Tapped += OnTapped;
            view.GestureRecognizers.Add ( tapRecognizer );
        }

        protected void override OnDetachingFrom ( Ver ver )
        {
            base .OnDetachingFrom ( vista );

            view.GestureRecognizers.Remove ( tapRecognizer );
            tapRecognizer.Tapped -= OnTapped;
        }

        vacío OnTapped ( objeto remitente, EventArgs args )
        {
            != IsToggled IsToggled;
        }
    }
}
```

los **ToggleBehavior** clase define una propiedad, lo que significa que no se puede compartir una **ToggleBehavior** en un Estilo.

He aquí una sencilla aplicación. los **ToggleLabel** agregados del programa **ToggleBehavior** a una Etiqueta y utiliza el **IsToggled** propiedad con una **DataTrigger** para cambiar el texto de la Etiqueta entre “pausa” y “Reproducción”, tal vez por una aplicación de música:

```
< Página de contenido xmlns = "http://xamarin.com/schemas/2014/forms"
    xmlns : X = "Http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns : kit de herramientas =
        "Cl-espacio de nombres: Xamarin.FormsBook.Toolkit; montaje = Xamarin.FormsBook.Toolkit"
```



```

        Tamaño de fuente = "Large"
        FontAttributes = "Negrita" />

        < Lapsos Texto = "Reproducción"
        Tamaño de fuente = "Pequeño" />
    </ FormattedString.Spans >
</ FormattedString >
</ Label.FormattedText >

< Label.Triggers >
    < DataTrigger Tipo de objetivo = "Etiqueta"
        Unión = "{ Unión Fuente = { X : Referencia toggleBehavior },
                    Camino = IsToggled}"
        Valor = "True">
        < Setter Propiedad = "FormattedText">
            < Setter.Value >
                < FormattedString >
                    < FormattedString.Spans >
                        < Lapsos Texto = "Pause"
                            Tamaño de fuente = "Pequeño" />

                            < Lapsos Texto = "/ Jugar"
                                Tamaño de fuente = "Large"
                                FontAttributes = "Negrita" />
                            </ FormattedString.Spans >
                        </ FormattedString >
                    </ Setter.Value >
                </ Setter >
            </ DataTrigger >
        </ Label.Triggers >
    </ Etiqueta >
</ Marco >

    < Etiqueta X : Nombre = "EventLabel"
        Texto = ""
        Tamaño de fuente = "Large"
        Opacidad = "0"
        HorizontalOptions = "Centro"
        VerticalOptions = "CenterAndExpand" />
    </ StackLayout >
</ Página de contenido >
```

los ToggleBehavior está unido a la Marco, y el Marco contiene una Etiqueta. (Observe que el Color de fondo del Marco se establece en Transparente en lugar del valor por defecto de nulo. Esto es necesario para atrapar a los eventos de TAP en las plataformas de Windows en tiempo de ejecución.)

Este programa muestra una forma de resolver un problema común con los botones de selección: ¿El texto (o ícono) se refiere a un estado o una acción? los Etiqueta aquí deja claro al mostrar el texto “En pausa / Jugar” pero con la palabra “pausa” más grande que la palabra “Reproducción”. Cuando el IsToggled propiedad es

Cierto, el DataTrigger Los cambios que la pantalla de modo que la palabra “Jugar” es más grande que la palabra “pausa”.

los PropertyChanged en el caso ToggleBehavior se maneja en el archivo de código subyacente:

```


pública clase parcial FormattedTextTogglePage : Pagina de contenido


```

```

{
    p
```

```

    p
```

```

    InitializeComponent ();
```

```

}
```

```

vacio OnBehaviorPropertyChanged ( o
```

```

    Si (Args.PropertyName == "IsToggled")
    {
        eventLabel.Text = "IsToggled = " + (( ToggleBehavior ) Remitente).IsToggled;
        eventLabel.Opacity = 1;
        eventLabel.FadeTo (0, 1000);
    }
}
}
```

los OnBehaviorPropertyChanged controlador comprueba si hay un cambio en la propiedad denominada "IsToggled". Tenga en cuenta que la remitente argumento para el controlador de eventos no es el elemento visual cuya grifos están siendo detectada (que es el Marco) pero el ToggleBehavior sí mismo. El código establece la Texto propiedad de la Etiqueta en la parte inferior de la página y establece el Opacidad a 1, pero luego se desvanece a cabo en el transcurso de un segundo para dar una sensación de un tiro de eventos:



Si te gusta la idea de definir los elementos visuales de una vista de palanca en XAML, pero prefieres un poco más estructura, la **Xamarin.FormsBook.Toolkit** biblioteca tiene una clase llamada **ToggleBase** que deriva de **Contenido**-**Ver e incorpora** **ToggleBehavior**. El constructor añade la **ToggleBehavior** al tamiento IORS colección de la clase y luego se conecta un controlador de eventos a la misma. La clase también define una **toggled** evento y su propia **IsToggled** propiedad que dispara este caso:

```

espacio de nombres Xamarin.FormsBook.Toolkit
{
    clase pública ToggleBase : ContentView
    {
        evento público Controlador de eventos < ToggledEventArgs > Biestado;

        sólo lectura estática pública BindableProperty IsToggledProperty =
            BindableProperty.Crear("IsToggled", tipo de ( bool ), tipo de ( ToggleBase ), falso ,
                BindingMode.TwoWay,
                PropertyChanged: (enlazable, oldValue, newValue) =>
                {
                    Controlador de eventos < ToggledEventArgs > Handler = (( ToggleBase ) Enlazable).Toggled;
                    Si (Manejador! = nulo )
                        Handler (enlazable, nuevo ToggledEventArgs (bool )nuevo valor);
                });
    }

    público ToggleBase ()
    {
        ToggleBehavior toggleBehavior = nuevo ToggleBehavior ();
        toggleBehavior.PropertyChanged += OnToggleBehaviorPropertyChanged;
        Behaviors.Add (toggleBehavior);
    }

    public bool IsToggled
    {
        conjunto {EstablecerValor (IsToggledProperty, valor );}
        obtener {regreso ( bool ) GetValue (IsToggledProperty);}
    }

    protected void OnToggleBehaviorPropertyChanged ( objeto remitente,
                                                    PropertyChangedEventArgs args)
    {
        Si (Args.PropertyName == "IsToggled")
        {
            IsToggled = (( ToggleBehavior ) Remitente).IsToggled;
        }
    }
}
}

```

los ToggleBase clase define toda la lógica de una vista de palanca sin los efectos visuales. En verdad, no se requiere la ToggleBehaviors clase. Se podría instalar su propia TapGestureRecognizer, pero el resultado sería básicamente la misma.

Puede crear una instancia del ToggleBase clase en un archivo XAML y el suministro de los elementos visuales como el contenido de la ToggleBase. Aquí hay un programa llamado **TraditionalCheckBox** que utiliza dos caracteres Unicode para una casilla sin marcar y una casilla marcada, similar a la Caja vistas en los capítulos 15 y 16:

```

< Pagina de contenido xmlns = "http://xamarin.com/schemas/2014/forms"
    xmlns : X = "Http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns : kit de herramientas =
        "Clr-espacio de nombres: Xamarin.FormsBook.Toolkit; montaje = Xamarin.FormsBook.Toolkit"
    X : Clase = "TraditionalCheckBox.TraditionalCheckBoxPage">
```

```

< StackLayout >
    < Kit de herramientas : ToggleBase X : Nombre = "Casilla de verificación" 
        HorizontalOptions = "Centro"
        VerticalOptions = "CenterAndExpand"
        toggled = "OnToggleBaseToggled">

        < StackLayout Orientación = "Horizontal" >
            < Etiqueta Texto = "& # X2610;" 
                Tamaño de fuente = "Large">
                < Label.Triggers >
                    < DataTrigger Tipo de objetivo = "Etiqueta" 
                        Unión = "{ Unión Fuente = { X : Referencia caja },
                            Camino = IsToggled}" 
                        Valor = "True">
                        < Setter Propiedad = "Texto" Valor = "& # X2611;" />
                    </ DataTrigger >
                </ Label.Triggers >
            </ Etiqueta >

            < Etiqueta Texto = "Cursiva texto" 
                Tamaño de fuente = "Grande" />
        </ StackLayout >
    </ Kit de herramientas : ToggleBase >

    < Etiqueta Texto = "Texto de ejemplo poner en cursiva" 
        Tamaño de fuente = "Large"
        HorizontalOptions = "Centro"
        VerticalOptions = "CenterAndExpand">
        < Label.Triggers >
            < DataTrigger Tipo de objetivo = "Etiqueta" 
                Unión = "{ Unión Fuente = { X : Referencia caja },
                    Camino = IsToggled}" 
                Valor = "True">
                < Setter Propiedad = "FontAttributes" Valor = "Cursiva" />
            </ DataTrigger >
        </ Label.Triggers >
    </ Etiqueta >

    < Etiqueta X : Nombre = "EventLabel" 
        Texto = ""
        Tamaño de fuente = "Large"
        Opacidad = "0"
        HorizontalOptions = "Centro"
        VerticalOptions = "CenterAndExpand" />

    </ StackLayout >
</ Página de contenido >

```

El archivo utiliza el XAML `IsToggled` propiedad como la fuente de dos enlaces de datos muy similares, cada uno dentro de una `DataTrigger`. En ambos casos, el `Fuente` propiedad se establece en el `ToggleBase` ejemplo, y el `Camino` propiedad se establece en el `IsToggled` propiedad de `ToggleBase`. El primero `DataTrigger` alterna entre la caja vacía y la casilla marcada para indicar el estado de la palanca, y el segundo `DataTrigger` en cursiva un texto cuando el Caja se puede cambiar entre el.

además, el toggled caso de ToggleBase se maneja en el archivo de código subyacente con un fundido de salida

Etiqueta:

```
pública clase parcial TraditionalCheckBoxPage : Pagina de contenido
{
    pública TraditionalCheckBoxPage ()
    {
        InitializeComponent ();
    }

    vacío OnToggleBaseToggled ( objeto remitente, ToggledEventArgs args )
    {
        eventLabel.Text = "IsToggled =" + Args.Value;
        eventLabel.Opacity = 1;
        eventLabel.FadeTo ( 0, 1000 );
    }
}
```

Aquí está el resultado:



Si necesita varias instancias de un tipo particular de vista de palanca, puede encapsular los elementos visuales en una clase que deriva de ToggleBase.

El siguiente ejemplo se deriva de ToggleBase para hacer una vista que es muy parecido a los Xamarin.Forms Cambiar, excepto con imágenes creadas por completo en XAML. Este "clon interruptor" se realiza con un poco BoxView que se mueve hacia adelante y hacia atrás en una Marco. Para la ejecución de la animación, el **Xamarin.FormsBook.Toolkit** biblioteca incluye una TranslateAction clase con propiedades que proporcionan argumentos para una llamada a Traducir a:

espacio de nombres **Xamarin.FormsBook.Toolkit**

```
{
    clase pública TranslateAction : TriggerAction < VisualElement >
    {
        público TranslateAction ()
        {
            // Configurar valores predeterminados.
            Longitud = 250;
            aliviar = aliviar .Lineal;
        }

        pública doble X{ conjunto ; obtener ; }

        pública doble Y{ conjunto ; obtener ; }

        public int Longitud { conjunto ; obtener ; }

        [ TypeConverter ( tipo de ( EasingConverter ))]
        público aliviar aliviar { conjunto ; obtener ; }

        protegido override void Invocar( VisualElement visual)
        {
            visual.TranslateXYTo (X, Y, ( uint ) Longitud, aliviando);
        }
    }
}
```

los **SwitchClone** clase que imita la Cambiar es parte del **SwitchCloneDemo** proyecto. Es totalmente hecho en XAML. El elemento raíz es la clase base de **ToggleBase**, y el **x: Class** atributo indica la clase derivada de **SwitchClone**. los recursos diccionario define varias constantes que permiten visuales que no son demasiado grandes, pero aún lo suficientemente grande como para ser un objetivo adecuado contacto:

```
< kit de herramientas : ToggleBase
    xmlns = "http://xamarin.com/schemas/2014/forms"
    xmlns : X = "Http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns : kit de herramientas =
        "clr-espacio de nombres: Xamarin.FormsBook.Toolkit; montaje = Xamarin.FormsBook.Toolkit"
    X : Clase = "SwitchCloneDemo.SwitchClone"
    X : Nombre = "Cambiar">

    < kit de herramientas : ToggleBase.Resources >
        < ResourceDictionary >
            < x: Doble X : Llave = "Altura" > 20 </ x: Doble >
            < x: Doble X : Llave = "Anchura" > 50 </ x: Doble >
            < x: Doble X : Llave = "Media anchura" > 2 5 </ x: Doble >
        </ ResourceDictionary >
    </ kit de herramientas : ToggleBase.Resources >

    < Marco Relleno = "2"
        OutlineColor = "Accent"
        Color de fondo = "Transparent">
        < AbsoluteLayout WidthRequest = "{ StaticResource anchura }" >
            < BoxView Color = "Accent"
                WidthRequest = "{ StaticResource medio ancho }"
                HeightRequest = "{ StaticResource altura }" >
```

```

< BoxView.Triggers >
  < DataTrigger Tipo de objetivo = "BoxView"
    Unión = "{ Unión Fuente = { X : Referencia palanca },
      Camino = IsToggled}"
    Valor = "True">
    < DataTrigger.EnterActions >
      < kit de herramientas : TranslateAction X = "[ StaticResource medio ancho ]"
        Longitud = "100" />
    </ DataTrigger.EnterActions >

    < DataTrigger.ExitActions >
      < kit de herramientas : TranslateAction Longitud = "100" />
    </ DataTrigger.ExitActions >
  </ DataTrigger >
</ BoxView.Triggers >
</ BoxView >
</ AbsoluteLayout >
</ Marco >
</ kit de herramientas : ToggleBase >

```

Tenga en cuenta que el elemento raíz tiene un nombre de "cambiar". Esto permite que el enlace de datos en el **DataTrigger** sobre el **BoxView** para hacer referencia al **IsToggled** propiedad definida por el **ToggleBase** clase. los **DataTrigger** no incluye una **Setter** sino que utiliza **EnterActions** y **ExitActions** para invocar la **TranslateAction** para desplazar el **BoxView** de ida y vuelta.

El archivo de código subyacente para **SwitchClone** no tiene más que una **InitializeComponent** llama, pero si usted necesita otras propiedades (por ejemplo, para el color o algún texto que la acompaña) se puede definir allí.

Al menos esa es la forma en que se codificó originalmente. Más tarde, el programa se negó a construir sobre las plataformas de Windows en tiempo de ejecución. Tal vez el problema tenía que ver con el elemento raíz del archivo XAML referencia a una clase en una biblioteca. En cualquier caso, una versión de sólo código de la clase hizo el trabajo, y este es el que viene incluido con el código de ejemplo para este capítulo:

```

clase SwitchClone : ToggleBase
{
  const doble height = 20;
  const doble width = 50;
  const doble de media anchura = 25;

  público SwitchClone ()
  {
    BoxView boxView = nuevo BoxView
    {
      color = Color .Acento,
      WidthRequest = media anchura,
      HeightRequest = altura
    };

    DataTrigger DataTrigger = nuevo DataTrigger ( tipo de ( BoxView ))
    {
      = vinculante nuevo Unión ( "IsToggled" , fuente: esta ),
      valor = cierto ,
    };
  }
}

```

los `SwitchCloneDemoPage` Clase muestra cuatro de estos clones cambiar consecutivas:

```
< Pagina de contenido xmlns = "http://xamarin.com/schemas/2014/forms"
    xmlns : X = "Http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns : local = "CLR-espacio de nombres: SwitchCloneDemo"
    X : Clase = "SwitchCloneDemo.SwitchCloneDemoPage">

    < Cuadricula VerticalOptions = "Center">
        < local : SwitchClone Grid.Column = "0"
            HorizontalOptions = "Center" />

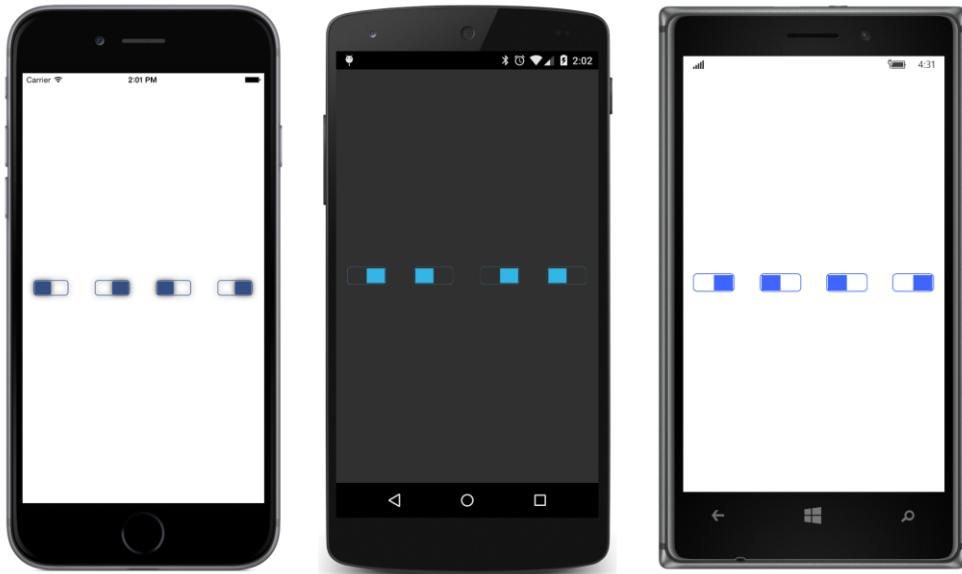
        < local : SwitchClone Grid.Column = "1"
            HorizontalOptions = "Center" />

        < local : SwitchClone Grid.Column = "2"
            HorizontalOptions = "Center" />

        < local : SwitchClone Grid.Column = "3"
            HorizontalOptions = "Center" />

    </ Cuadricula >
</ Pagina de contenido >
```

Y aquí están:



Por supuesto, una vez que comience a pensar en el uso de animaciones, es posible empezar a recibir algunas ideas interesantes (o tal vez francamente impares) de lo que es una vista de palanca podría ser similar. Para darle algunas opciones más, aquí hay una RotateAction clase:

```
espacio de nombres Xamarin.FormsBook.Toolkit
{
    clase pública RotateAction : TriggerAction < VisualElement >
    {
        público RotateAction ()
        {
            // Configurar valores predeterminados.
            ancla = nuevo Punto (0.5, 0.5);
            Rotación = 0;
            Longitud = 250;
            aliviar = aliviar .Lineal;
        }

        público Punto ancla { conjunto ; obtener ; }

        pública doble rotación { conjunto ; obtener ; }

        public int Longitud { conjunto ; obtener ; }

        [TypeConverter ( tipo de ( EasingConverter ))]
        público aliviar aliviar { conjunto ; obtener ; }

        protected void override Invocar( VisualElement visual )
        {
            visual.AnchorX = Anchor.X;
            visual.AnchorY = Anchor.Y;
            visual.RotateTo (Rotación, ( uint ) Longitud, aliviando);
        }
    }
}
```

1

Los **LeverToggle** programa tiene un archivo XAML que se dedica a un solo interruptor de palanca construido a partir de dos BoxView elementos. El primero BoxView se asemeja a una base para la segunda, que funciona como una palanca. Observe que el DataTrigger en el segundo BoxView contiene una Setter para cambiar el color de la BoxView tanto como EnterActions y ExitActions para invocar animaciones que se mueven la palanca hacia atrás y hacia adelante:

```

< Pagina de contenido xmlns = "http://xamarin.com/schemas/2014/forms"
    xmlns : X = "Http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns : kit de herramientas =
        "Clr-espacio de nombres: Xamarin.FormsBook.Toolkit; montaje = Xamarin.FormsBook.Toolkit"
    X : Clase = "LeverToggle.LeverTogglePage">

    < kit de herramientas : ToggleBase X : Nombre = "Cambiar"
        HorizontalOptions = "Centro"
        VerticalOptions = "Center">

        < AbsoluteLayout >
            < BoxView Color = "Gris"
                AbsoluteLayout.LayoutBounds = "0, 75, 100, 25">
                < BoxView.Triggers >
                    < DataTrigger Tipo de objetivo = "BoxView"
                        Unión = "{ Unión Fuente = { X : Referencia palanca },
                            Camino = IsToggled}"
                        Valor = "True">
                        < Setter Propiedad = "Color" Valor = "Lime" />
                    </ DataTrigger >
                </ BoxView.Triggers >
            </ BoxView >

            < BoxView Color = "Gris"
                AbsoluteLayout.LayoutBounds = "45, 0, 10, 100"
                anchorX = "0.5"
                anchorY = "1"
                Rotación = "- 30">
                < BoxView.Triggers >
                    < DataTrigger Tipo de objetivo = "BoxView"
                        Unión = "{ Unión Fuente = { X : Referencia palanca },
                            Camino = IsToggled}"
                        Valor = "True">
                        < Setter Propiedad = "Color" Valor = "Lime" />
                    </ DataTrigger >
                </ BoxView.Triggers >
            </ BoxView >

            < DataTrigger.EnterActions >
                < kit de herramientas : RotateAction Ancla = "0.5, 1" Rotación = "30" />
            </ DataTrigger.EnterActions >

            < DataTrigger.ExitActions >
                < kit de herramientas : RotateAction Ancla = "0.5, 1" Rotación = "- 30" />
            </ DataTrigger.ExitActions >
        </ DataTrigger >
    </ AbsoluteLayout >
</ Pagina de contenido >

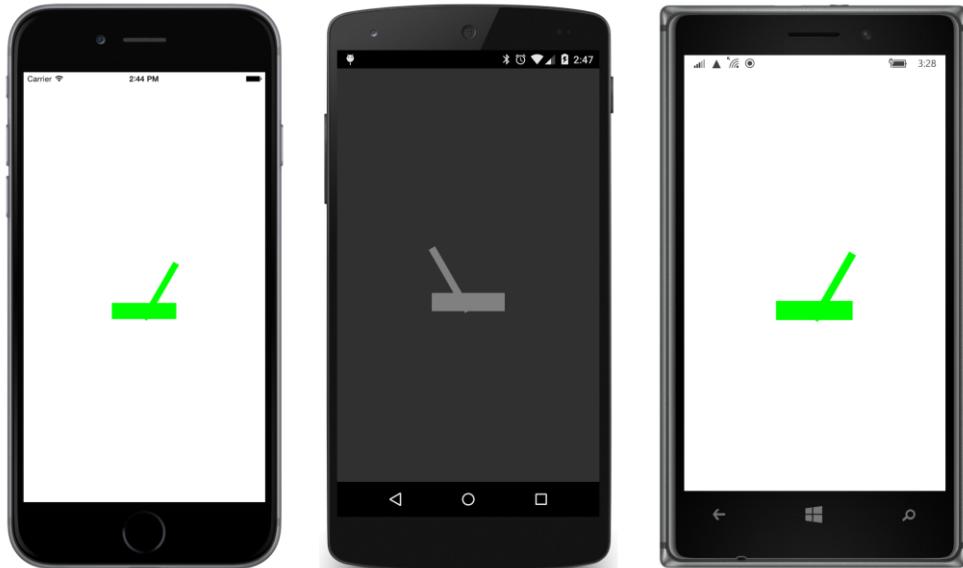
```

```

</ BoxView.Triggers >
</ BoxView >
</ AbsoluteLayout >
<kit de herramientas : ToggleBase >
<Pagina de contenido >

```

El estado untoggled se muestra en la pantalla de Android, mientras que las 10 pantallas móviles iOS y Windows muestran el estado conmutado:



Responder a los toques

Las diversas manifestaciones de vistas de palanca demuestran una manera de responder a los toques dentro de un archivo XAML. Si los eventos de derivación se integraron en el `VisualElement` clase, se podría llegar a ellos más directamente y con mayor facilidad usando `EventTrigger`. Pero no se puede adjuntar una `EventTrigger` a una `TapGestureRecognizer`.

ognizer.

Moverse por esa pequeña restricción es el propósito de un comportamiento dedicada exclusivamente a un grifo. Se llama `TapBehavior`:

```

espacio de nombres Xamarin.FormsBook.Toolkit
{
    clase pública TapBehavior : Comportamiento <Ver >
    {
        TapGestureRecognizer tapGesture;

        estático solo lectura BindablePropertyKey IsTriggeredKey =
            BindableProperty.CreateReadonly( "IsTriggered" , tipo de ( bool ),
                tipo de ( TapBehavior ), falso );
    }
}

```

```
sólo lectura estática pública BindableProperty IsTriggeredProperty =  
IsTriggeredKey.BindableProperty
```

Los TapBehavior clase define una propiedad booleana llamada IsTriggered, pero no funciona exactamente como una característica normal. Por un lado, está respaldado por una propiedad enlazable de sólo lectura. Esto significa que el IsTriggered propiedad se puede establecer sólo dentro de la TapBehavior clase, y la única vez que los conjuntos de clases IsTriggered está en el controlador de eventos para el TapGestureRecognizer, cuando el IsTrig-
Gered propiedad se convierte cierto por sólo una décima de segundo.

En otras palabras, el aprovechado evento se convierte en un muy breve pico de una propiedad de valor algo que recuerda cómo los eventos se desencadenan en el hardware digital. Pero el IsTriggered propiedad, entonces se puede hacer referencia en una DataTrigger.

Supongamos que te gusta la idea de la ShiverButton, pero deseas aplicar el concepto a algo que no sea una Botón, lo que significa que necesitas responder a eventos. No se puede utilizar una EventTrigger, pero el TapBehavior le permite utilizar un DataTrigger en lugar.

Para demostrarlo, aquí está **BoxViewTapShiver**, que se une TapBehavior objetos a tres Caja-

ShiverAction en su EnterActions colección:

```

< Pagina de contenido xmlns = "http://xamarin.com/schemas/2014/forms"
    xmlns : X = "Http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns : kit de herramientas =
        "Clr-espacio de nombres: Xamarin.FormsBook.Toolkit; montaje = Xamarin.FormsBook.Toolkit"
    X : Clase = "BoxViewTapShiver.BoxViewTapShiverPage">

    < ContentPage.Resources >
        < ResourceDictionary >
            < Estilo Tipo de objetivo = "BoxView">
                < Setter Propiedad = "WidthRequest" Valor = "200" />
                < Setter Propiedad = "HeightRequest" Valor = "50" />
                < Setter Propiedad = "HorizontalOptions" Valor = "Center" />
                < Setter Propiedad = "VerticalOptions" Valor = "CenterAndExpand" />
            </ Estilo >
        </ ResourceDictionary >
    </ ContentPage.Resources >

    < StackLayout >
        < BoxView Color = "Red">
            < BoxView.Behaviors >
                < kit de herramientas : TapBehavior X : Nombre = "TapBehavior1" />
            </ BoxView.Behaviors >

            < BoxView.Triggers >
                < DataTrigger Tipo de objetivo = "BoxView"
                    Unión = "{ Unión Fuente = { X : Referencia tapBehavior1 },
                    Camino = IsTriggered}"
                    Valor = "True">
                    < DataTrigger.EnterActions >
                        < kit de herramientas : ShiverAction />
                    </ DataTrigger.EnterActions >
                </ DataTrigger >
            </ BoxView.Triggers >
        </ BoxView >

        < BoxView Color = "Green">
            < BoxView.Behaviors >
                < kit de herramientas : TapBehavior X : Nombre = "TapBehavior2" />
            </ BoxView.Behaviors >

            < BoxView.Triggers >
                < DataTrigger Tipo de objetivo = "BoxView"
                    Unión = "{ Unión Fuente = { X : Referencia tapBehavior2 },
                    Camino = IsTriggered}"
                    Valor = "True">
                    < DataTrigger.EnterActions >
                        < kit de herramientas : ShiverAction />
                    </ DataTrigger.EnterActions >
                </ DataTrigger >
            </ BoxView.Triggers >
        </ BoxView >
    
```

```

< BoxView Color = "Blue">
  < BoxView.Behaviors >
    < kit de herramientas : TapBehavior X : Nombre = "TapBehavior3" />
  </ BoxView.Behaviors >

  < BoxView.Triggers >
    < DataTrigger Tipo de objetivo = "BoxView"
      Unión = "[ Unión Fuente = { X : Referencia tapBehavior3 },
      Camino = IsTriggered}"
      Valor = "True">
      < DataTrigger.EnterActions >
        < kit de herramientas : ShiverAction />
      </ DataTrigger.EnterActions >
    </ DataTrigger >
  </ BoxView.Triggers >
</ BoxView >
</ StackLayout >
</ Página de contenido >

```

Cada uno de los tres TapBehavior objetos tiene un nombre único, que hace referencia la correspondiente

DataTrigger. Cuando puntee en un BoxView, se estremece, y todos ellos trabajan de manera independiente.

Es muy tentador para poner el TapBehavior y DataTrigger objetos en una Estilo a reducir el margen de beneficio repetitiva, pero eso no va a funcionar. Eso provocaría una sola TapBehavior que se repartirán entre los tres BoxView elementos. Además, cada DataTrigger se refiere a un correspondiente Grifo-

Comportamiento por nombre.

Si desea reducir el margen de beneficio en este caso, tendrá una vez más, es necesario definir una nueva clase. Los **ShiverViews** programa demuestra esto. En primer lugar, define una clase denominada ShiverView que deriva de BoxView y añade el TapBehavior y DataTrigger:

```

< BoxView xmlns = "http://xamarin.com/schemas/2014/forms"
  xmlns : X = "http://schemas.microsoft.com/winfx/2009/xaml"
  xmlns : kit de herramientas =
  "Clr-espacio de nombres: Xamarin.FormsBook.Toolkit; montaje = Xamarin.FormsBook.Toolkit"
  X : Clase = "ShiverViews.ShiverView">

  < BoxView.Behaviors >
    < kit de herramientas : TapBehavior X : Nombre = "TapBehavior" />
  </ BoxView.Behaviors >

  < BoxView.Triggers >
    < DataTrigger Tipo de objetivo = "BoxView"
      Unión = "[ Unión Fuente = { X : Referencia tapBehavior },
      Camino = IsTriggered}"
      Valor = "True">
      < DataTrigger.EnterActions >
        < kit de herramientas : ShiverAction />
      </ DataTrigger.EnterActions >
    </ DataTrigger >
  </ BoxView.Triggers >
</ BoxView >

```

Al igual que con la `SwitchClone` clase, también podría añadir algunas propiedades en el archivo de código subyacente y hacer referencia a ellos en el archivo XAML.

Los `ShiverViewsPage` archivo XAML puede entonces sólo una instancia de tres independientes `ShiverView` los objetos con un estilo implícita:

```
< Pagina de contenido xmlns = "http://xamarin.com/schemas/2014/forms"
    xmlns : X = "Http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns : local = "clr-namespace: ShiverViews"
    X : Clase = "ShiverViews.ShiverViewsPage">

    < StackLayout >
        < StackLayout.Resources >
            < ResourceDictionary >
                < Estilo Tipo de objetivo = "Local: ShiverView" >
                    < Setter Propiedad = "WidthRequest" Valor = "200" />
                    < Setter Propiedad = "HeightRequest" Valor = "50" />
                    < Setter Propiedad = "HorizontalOptions" Valor = "Center" />
                    < Setter Propiedad = "VerticalOptions" Valor = "CenterAndExpand" />
                </ Estilo >
            </ ResourceDictionary >
        </ StackLayout.Resources >

        < local : ShiverView Color = "Red" />
        < local : ShiverView Color = "Green" />
        < local : ShiverView Color = "Blue" />
    </ StackLayout >
</ Pagina de contenido >
```

Botones de radio

Los radios integradas en los tableros de instrumentos de automóviles antiguos a menudo aparece una fila de botones de media docena (más o menos) que podrían ser “programados” para varias estaciones de radio. Empujar en uno de estos botones causó la radio para saltar a esa estación preseleccionada, y también causó el botón de la selección previa para que salga.

Esas radios para automóviles viejos son ahora las antigüedades, pero las opciones que se excluyen mutuamente en nuestras pantallas de computadoras todavía están representados por objetos visuales que llamamos *botones de radio*.

Los botones de radio son algo similares a alterna o casillas de verificación. Sin embargo, botones de radio se encuentran siempre en un grupo de dos o más. Seleccionar o comprobar cualquier botón en ese grupo hace que los demás se vuelven a marcar.

La lógica detrás de los botones de radio es complicada debido a una aplicación podría presentar varios grupos de botones de radio en la misma página, y estos grupos debe funcionar de manera independiente. Al pulsar un botón en un grupo sólo debe afectar a los otros botones dentro del grupo, y no con los botones en cualquier otro grupo.

Tradicionalmente, los botones de radio se agruparon con un parente común. En la terminología Xamarin.Forms, botones de radio que son hijos de un mismo `StackLayout` se considera que son del mismo grupo, mientras

botones de radio que son hijos de otra StackLayout están en otro grupo independiente.

Sin embargo, hay una manera más generalizada para distinguir grupos de botones de radio, y que está dando a cada grupo un nombre único, que en realidad significa que cada botón de opción dentro de ese grupo hace referencia al mismo nombre.

El problema con estos nombres es que añaden un poco de sobrecarga adicional, sobre todo cuando se necesita sólo un grupo de botones de radio. Por esa razón, debe haber una asignación para un grupo de botones de radio que es *no* identificado por un nombre. Esto se llama el *defecto* grupo.

Aquí es una RadioBehavior clase en el **Xamarin.FormsBook.Toolkit** biblioteca que se basa en esos principios. Adjuntar este comportamiento a todas las vistas que desea convertir en un botón de radio. Como el **ToggleBehavior** clase, **RadioBehavior** establece una TapGestureRecognizer en el elemento visual a la que se adjunta. No define una **IsToggled** propiedad como **ToggleBehavior**, pero sí definir una **Está chequeado** propiedad que es bastante similar e indica si el botón de opción está activada o desactivada. Los **RadioBehavior** clase también define una **Nombre del grupo** propiedad de tipo cuerda para identificar el grupo; un nulo valor o una cadena vacía indica el grupo predeterminado.

Los **RadioBehavior** clase necesita para almacenar todos los botones de radio instanciados por grupo, por lo que define dos colecciones estáticas, uno de los cuales es un simple **Lista <RadioBehavior>** para todos los objetos del grupo por defecto, y la otra es una Diccionario con una tecla correspondiente al nombre del grupo que hace referencia a una

Lista **<RadioBehavior>** colección para todos los objetos de ese grupo denominado:

```
espacio de nombres Xamarin.FormsBook.Toolkit
{
    clase pública RadioBehavior : Comportamiento < Ver >
    {
        TapGestureRecognizer tapRecognizer;
        estático Lista < RadioBehavior > = DefaultGroup nuevo Lista < RadioBehavior > ();
        estático Diccionario < cuerda , Lista < RadioBehavior >> RadioGroups =
            nuevo Diccionario < cuerda , Lista < RadioBehavior >> ();

        público RadioBehavior ()
        {
            defaultGroup.Add ( esta );
        }

        sólo lectura estática pública BindableProperty IsCheckedProperty =
            BindableProperty .Crear( "Está chequeado" ,
                tipo de ( bool ),
                tipo de ( RadioBehavior ),
                falso ,
                PropertyChanged: OnIsCheckedChanged);

        public bool Está chequeado
        {
            conjunto { EstablecerValor ( IsCheckedProperty , valor ); }
            obtener { regreso ( bool ) GetValue ( IsCheckedProperty ); }
        }

        hoyo estatico OnIsCheckedChanged ( bindableObject enlazable , objeto valor antiguo ,
            bindableObject nuevo valor , EventArgs args )
        {
            if ( enlazable == null )
                return;
            if ( enlazable != null )
                enlazable.SetValue ( IsCheckedProperty , nuevo valor );
        }
    }
}
```

```

        objeto nuevo valor)
{
    RadioBehavior comportamiento = ( RadioBehavior ) Enlazable;

    Si (( bool )nuevo valor)
    {
        cuerda nombre_de_grupo = behavior.GroupName;
        Lista < RadioBehavior > = comportamientos nulo ;

        Si ( Cuerda .IsNullOrEmpty (nombre_de_grupo))
        {
            comportamientos = DefaultGroup;
        }
        más
        {
            comportamientos = RadioGroups [nombre_de_grupo];
        }

        para cada ( RadioBehavior otherBehavior en comportamientos)
        {
            Si (OtherBehavior! = Comportamiento)
            {
                otherBehavior.IsChecked = falso ;
            }
        }
    }
}

sólo lectura estática pública BindableProperty GroupNameProperty =
BindableProperty .Crear( "Nombre del grupo" ,
                        tipo de ( cuerda ),
                        tipo de ( RadioBehavior ),
                        nulo ,
                        PropertyChanged: OnGroupNameChanged);

public string Nombre del grupo
{
    conjunto {EstablecerValor (GroupNameProperty, valor ); }
    obtener { regreso ( cuerda ) GetValue (GroupNameProperty); }
}

hoyo estatico OnGroupNameChanged ( bindableObject enlazable, objeto valor antiguo,
                                    objeto nuevo valor)
{
    RadioBehavior comportamiento = ( RadioBehavior ) Enlazable;
    cuerda oldGroupName = ( cuerda )valor antiguo;
    cuerda newGroupName = ( cuerda )nuevo valor;

    Si ( Cuerda .IsNullOrEmpty (oldGroupName))
    {
        // eliminar el comportamiento del grupo predeterminado.
        defaultGroup.Remove (comportamiento);
    }
    más
}

```

```
{  
    // Eliminar el RadioBehavior de la colección RadioGroups.  
    Lista < RadioBehavior > Comportamientos = RadioGroups [oldGroupName];  
    behaviors.Remove (comportamiento);  
  
    // Se puede olvidarse de la colección si está vacío.  
    Si (Behaviors.Count == 0)  
    {  
        radioGroups.Remove (oldGroupName);  
    }  
}  
  
Si ( Cuerda .IsNullOrEmpty (newGroupName))  
{  
    // Añadir el nuevo comportamiento al grupo predeterminado.  
    defaultGroup.Add (comportamiento);  
}  
más  
{  
    Lista < RadioBehavior > = comportamientos nulo ;  
  
    Si (RadioGroups.ContainsKey (newGroupName))  
    {  
        // Obtener el grupo nombrado.  
        comportamientos = RadioGroups [newGroupName];  
    }  
    más  
{  
        // Si no existe ese grupo, lo crea.  
        comportamientos = nuevo Lista < RadioBehavior > 0;  
        radioGroups.Add (newGroupName, comportamientos);  
    }  
  
    // Añadir el comportamiento al grupo.  
    behaviors.Add (comportamiento);  
}  
}  
  
protegido override void OnAttachedTo ( Ver ver)  
{  
    base .OnAttachedTo (vista);  
  
    tapRecognizer = nuevo TapGestureRecognizer ();  
    tapRecognizer.Tapped += OnTapRecognizerTapped;  
    view.GestureRecognizers.Add (tapRecognizer);  
}  
  
protegido override void OnDetachingFrom ( Ver ver)  
{  
    base .OnDetachingFrom (vista);  
  
    view.GestureRecognizers.Remove (tapRecognizer);  
    tapRecognizer.Tapped -= OnTapRecognizerTapped;  
}
```

```
vacío OnTapRecognizerTapped ( objeto remitente, EventArgs args)  
{  
    IsChecked = cierto ;  
}  
}  
}
```

los TapGestureRecognizer manejador en la parte inferior de la lista es muy simple: Cuando se toca el objeto visual, la RadioBehavior objeto unido a dicho objeto visual establece su Está chequeado propiedad a cierto. Si el Está chequeado propiedad era anteriormente falso, que el cambio hace una llamada a la

OnIsCheckedChanged método, que establece el Está chequeado la propiedad de toda la RadioBehavior objetos en el mismo grupo a falso.

He aquí una simple demostración de cierta lógica interactiva para seleccionar el tamaño de una camiseta. Los tres botones de radio son simples Etiqueta elementos con propiedades de texto de "Pequeño", "Medio", y "grande", y por eso el programa se llama Radiomarcadores. Cada Etiqueta tiene un RadioBehavior en su comportamientos colección. Cada RadioBehavior se da una x: Nombre para enlaces de datos, pero todos los RadioBehavior los objetos tienen un valor predeterminado Nombre del grupo del valor de la propiedad nulo. Cada Etiqueta También tiene una DataTrigger en su disparadores colección que se une a la correspondiente RadioBehavior para encender la Color de texto del Etiqueta a verde cuando el Está chequeado propiedad es cierto.

Observe que el `Está chequeado viviendas en el centro` RadioBehavior propiedad se inicializa a cierto

para seleccionar ese objeto cuando el programa se pone en marcha:

```
< Pagina de contenido xmlns = "http://xamarin.com/schemas/2014/forms"
    xmlns : X = "Http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns : kit de herramientas =
        "clr-espacio de nombres: Xamarin.FormsBook.Toolkit; montaje = Xamarin.FormsBook.Toolkit"
    xmlns : local = "clr-namespace: radiomarcadores"
    X : Clase = "RadioLabels.RadioLabelsPage"
    Relleno = "0, 50, 0, 0">
```

```
< StackLayout >
    < Cuadricula >
        < Grid.Resources >
            < ResourceDictionary >
                < Estilo Tipo de objetivo = "Etiqueta" >
                    < Setter Propiedad = "FontSize" Valor = "Medium" />
                    < Setter Propiedad = "HorizontalAlignment" Valor = "Center" />
                </ Estilo >
            </ ResourceDictionary >
        </ Grid.Resources >
    < Etiqueta Texto = "Pequeño" >
        Color de texto = "Gris"
        Grid.Column = "0"
    < Label.Behaviors >
        < kit de herramientas : RadioBehavior X : Nombre = "SmallRadio" />
    </ Label.Behaviors >
</ StackLayout >
```

```

< Label.Triggers >
  < DataTrigger Tipo de objetivo = "Etiqueta" 
    Unión = "{ Unión Fuente = { X : Referencia smallRadio },
      Camino = IsChecked}"
    Valor = "True">
    < Setter Propiedad = "TextColor" Valor = "Green" />
  </ DataTrigger >
</ Label.Triggers >
</ Etiqueta >

< Etiqueta Texto = "Medium" 
  Color de texto = "Gris"
  Grid.Column = "1">
< Label.Behaviors >
  < kit de herramientas : RadioBehavior X : Nombre = "MediumRadio" 
    Está chequeado = "True" />
</ Label.Behaviors >

< Label.Triggers >
  < DataTrigger Tipo de objetivo = "Etiqueta" 
    Unión = "{ Unión Fuente = { X : Referencia mediumRadio },
      Camino = IsChecked}"
    Valor = "True">
    < Setter Propiedad = "TextColor" Valor = "Green" />
  </ DataTrigger >
</ Label.Triggers >
</ Etiqueta >

< Etiqueta Texto = "Large" 
  Color de texto = "Gris"
  Grid.Column = "2">
< Label.Behaviors >
  < kit de herramientas : RadioBehavior X : Nombre = "LargeRadio" />
</ Label.Behaviors >

< Label.Triggers >
  < DataTrigger Tipo de objetivo = "Etiqueta" 
    Unión = "{ Unión Fuente = { X : Referencia largeRadio },
      Camino = IsChecked}"
    Valor = "True">
    < Setter Propiedad = "TextColor" Valor = "Green" />
  </ DataTrigger >
</ Label.Triggers >
</ Etiqueta >

< Cuadricula >
< Cuadricula VerticalOptions = "CenterAndExpand" 
  HorizontalOptions = "Center">

  < Imagen Fuente = "{ local : ImageResource radiomarcadores .Images.tee200.png }" 
    Es visible = "{ Unión Fuente = { X : Referencia smallRadio },
      Camino = IsChecked}" />
  < Imagen Fuente = "{ local : ImageResource radiomarcadores .Images.tee250.png }" 
    Es visible = "{ Unión Fuente = { X : Referencia mediumRadio },
      Camino = IsChecked}" />
</ Cuadricula >

```

```

        Camino = IsChecked)?"/>

<Imagen Fuente ="{ local : ImageResource radiomarcadores .Images.tee300.png}">
    Es visible = "{ Unión Fuente = { X : Referencia largeRadio },
        Camino = IsChecked})"/>

</Cuadricula>
</StackLayout>
</Pagina de contenido>

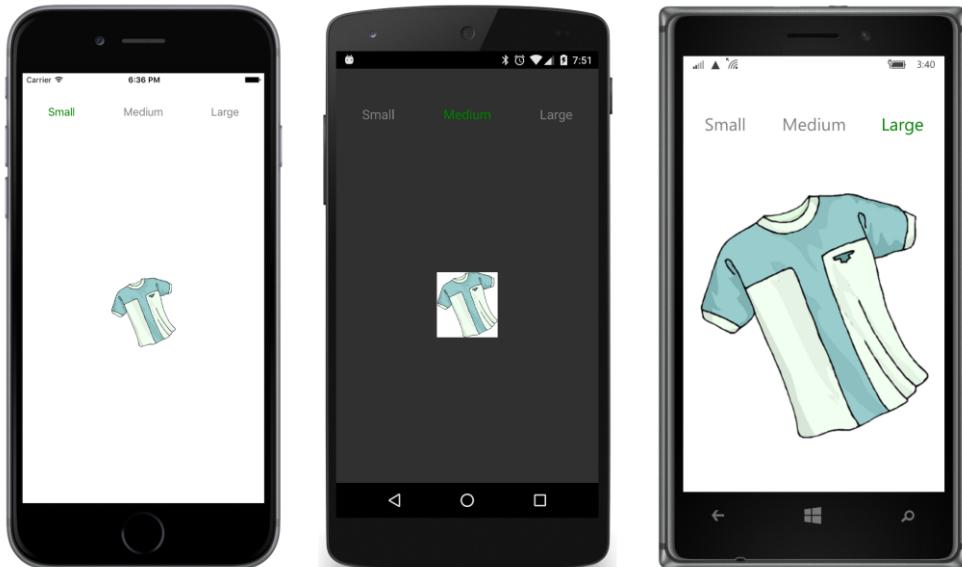
```

Otra complicación intrínseca a los botones de radio consiste en hacer uso de la opción seleccionada. En algunos casos se desea que cada botón de radio dentro de un grupo a ser representado por un miembro de la enumeración particular. (En este ejemplo, dicha enumeración podría tener tres miembros, nombrada Pequeño mediano, y

Grande.) La consolidación de un grupo de botones de opción en un valor de enumeración, obviamente, implica más código.

los radiomarcadores programa evita esos problemas y simplemente se une al Está chequeado propiedades de los tres RadioBehavior se opone a la Es visible propiedades de tres Imagen elementos que comparten un singlecell Cuadricula en la parte inferior del archivo XAML. Estos muestran un mapa de bits de tamaño diferente dependiendo de la selección.

Los tamaños relativos de estos mapas de bits no es tan evidente en estas capturas de pantalla porque cada plataforma muestra los mapas de bits en algo diferentes tamaños:



los DataTrigger unido a cada Etiqueta cambia el Color de texto de su color de estilo de gris a Verde cuando se selecciona ese elemento.

Si desea modificar varias propiedades de cada Etiqueta cuando se selecciona ese elemento, se puede añadir más Setter se opone a la DataTrigger. Sin embargo, un mejor enfoque consiste en consolidar el Setter objetos

en un Estilo, y luego hacer referencia a la Estilo en el DataTrigger.

Esto se demuestra en el **RadioStyle** programa. los recursos Inglés para la página define una Estilo con la tecla de “baseStyle” que define la apariencia de un sin marcar Etiqueta, y una Estilo con la tecla de “selectedStyle” que se basa en “baseStyle” sino que define la apariencia de una marcada Etiqueta. los recursos colección concluye con un estilo implícito para Etiqueta que es lo mismo que “baseStyle”:

```
< Pagina de contenido xmlns = "http://xamarin.com/schemas/2014/forms"
    xmlns : X = "Http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns : local = "CLR-espacio de nombres: RadioStyle"
    xmlns : kit de herramientas =
        "Clr-espacio de nombres: Xamarin.FormsBook.Toolkit; montaje = Xamarin.FormsBook.Toolkit"
    X : Clase = "RadioStyle.RadioStylePage"
    Relleno = "0, 50, 0, 0">

    < ContentPage.Resources >
        < ResourceDictionary >
            < Estilo X : Llave = "BaseStyle" Tipo de objetivo = "Etiqueta">
                < Setter Propiedad = "TextColor" Valor = "Gris" />
                < Setter Propiedad = "FontSize" Valor = "Pequeño" />
                < Setter Propiedad = "HorizontalTextAlignment" Valor = "Center" />
                < Setter Propiedad = "VerticalTextAlignment" Valor = "Center" />
            </ Estilo >

            < Estilo X : Llave = "SelectedStyle" Tipo de objetivo = "Etiqueta"
                Residencia en = "{ StaticResource baseStyle }">
                < Setter Propiedad = "TextColor" Valor = "Green" />
                < Setter Propiedad = "FontSize" Valor = "Medium" />
                < Setter Propiedad = "FontAttributes" Valor = "Negrita, cursiva" />
            </ Estilo >

            <! - estilo Implicito ->
            < Estilo Tipo de objetivo = "Etiqueta" Residencia en = "[ StaticResource baseStyle ]">
        </ ResourceDictionary >
    </ ContentPage.Resources >

    < StackLayout >
        < Cuadricula >
            < Etiqueta Texto = "Pequeño"
                Grid.Column = "0">
                < Label.Behaviors >
                    < kit de herramientas : RadioBehavior X : Nombre = "SmallRadio" />
                </ Label.Behaviors >

                < Label.Triggers >
                    < DataTrigger Tipo de objetivo = "Etiqueta"
                        Unión = "( Unión Fuente = { X : Referencia smallRadio },
                            Camino = IsChecked )"
                        Valor = "True">
                        < Setter Propiedad = "Estilo" Valor = "[ StaticResource selectedStyle ]">
                    </ DataTrigger >
                </ Label.Triggers >
            </ Etiqueta >
        </ Cuadricula >
    </ StackLayout >

```

```

</ Etiqueta >

< Etiqueta Texto = "Medium"
    Grid.Column = "1">
    < Label.Behaviors >
        < kit de herramientas : RadioBehavior X : Nombre = "MediumRadio"
            Está chequeado = "True" />
    </ Label.Behaviors >

    < Label.Triggers >
        < DataTrigger Tipo de objetivo = "Etiqueta"
            Unión = "{ Unión Fuente = { X : Referencia mediumRadio },
                        Camino = IsChecked}">
            Valor = "True">
            < Setter Propiedad = "Estilo" Valor = "{ StaticResource selectedStyle }"/>
        </ DataTrigger >
    </ Label.Triggers >
</ Etiqueta >

< Etiqueta Texto = "Large"
    Grid.Column = "2">
    < Label.Behaviors >
        < kit de herramientas : RadioBehavior X : Nombre = "LargeRadio" />
    </ Label.Behaviors >

    < Label.Triggers >
        < DataTrigger Tipo de objetivo = "Etiqueta"
            Unión = "{ Unión Fuente = { X : Referencia largeRadio },
                        Camino = IsChecked}">
            Valor = "True">
            < Setter Propiedad = "Estilo" Valor = "{ StaticResource selectedStyle }"/>
        </ DataTrigger >
    </ Label.Triggers >
</ Etiqueta >

</ Cuadrícula >

< Cuadrícula VerticalOptions = "CenterAndExpand"
    HorizontalOptions = "Center">

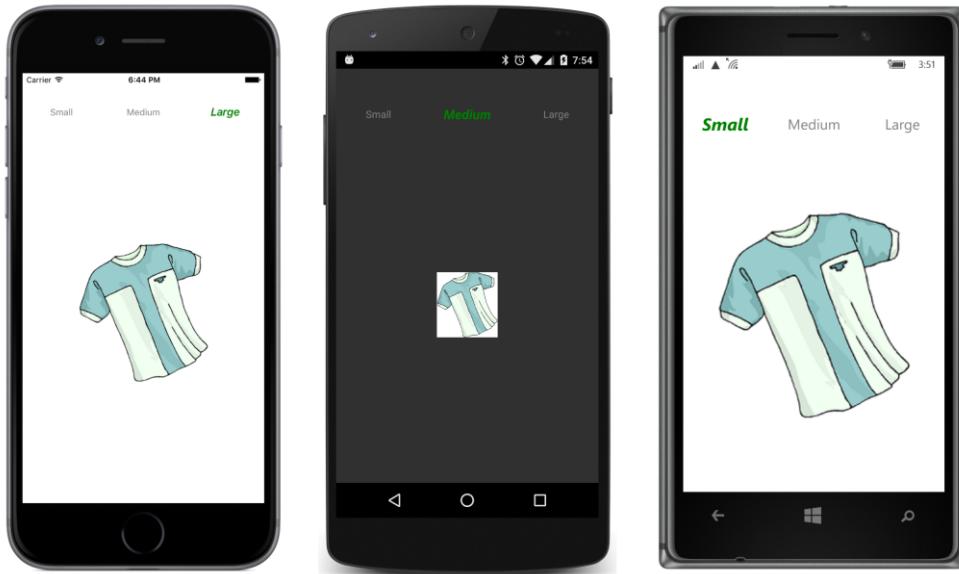
    < Imagen Fuente = "{ local : ImageResource RadioStyle .Images.tee200.png}"
        Es visible = "{ Unión Fuente = { X : Referencia smallRadio },
                        Camino = IsChecked}"/>

    < Imagen Fuente = "{ local : ImageResource RadioStyle .Images.tee250.png"
        Es visible = "{ Unión Fuente = { X : Referencia mediumRadio },
                        Camino = IsChecked}"/>

    < Imagen Fuente = "{ local : ImageResource RadioStyle .Images.tee300.png"
        Es visible = "{ Unión Fuente = { X : Referencia largeRadio },
                        Camino = IsChecked}"/>
</ Cuadrícula >
</ StackLayout >
</ Página de contenido >

```

Antes de este capítulo, Setter objetos sólo se encontraron en Estilo definiciones, por lo que podría parecer un poco extraño ver a una Setter objeto en el DataTrigger que establece el Estilo propiedad para el Etiqueta. Sin embargo, las imágenes demuestran que funciona bien. Ahora el elemento seleccionado está en una fuente grande con negrita y cursiva, además de un color diferente:



Usted también puede divertirse creando nuevos tipos de elementos visuales para identificar el elemento seleccionado en un grupo de botones de radio. los RadiolImages programa contiene cuatro mapas de bits que indican los diferentes modos de transporte. los Imagen elementos que hacen referencia a estos mapas de bits son cada uno de una ContentView al que está unido el RadioBehavior y una DataTrigger que cambia el color de la ContentView:

```
< Página de contenido xmlns = "http://xamarin.com/schemas/2014/forms"
      xmlns : X = "Http://schemas.microsoft.com/winfx/2009/xaml"
      xmlns : local = "clr-namespace: RadiolImages"
      xmlns : kit de herramientas =
          "Clr-espacio de nombres: Xamarin.FormsBook.Toolkit; montaje = Xamarin.FormsBook.Toolkit"
      X : Clase = "RadiolImages.RadiolImagesPage">

< ContentPage.Resources >
    < ResourceDictionary >
        < Estilo Tipo de objetivo = "ContentView" >
            < Setter Propiedad = "WidthRequest" Valor = "75" />
            < Setter Propiedad = "HeightRequest" Valor = "75" />
            < Setter Propiedad = "Relleno" Valor = "10" />
        </ Estilo >

        < Color X : Llave = "SelectedColor" > # 80C0FF < Color >
    </ ResourceDictionary >
</ ContentPage.Resources >

< StackLayout HorizontalOptions = "Inicio" >
```

```
VerticalOptions = "Centro"
Relleno = "20, 0"
Espaciado = "0">

< ContentView >
    < ContentView.Behaviors >
        < kit de herramientas : RadioBehavior X : Nombre = "PedestrianRadio" />
    </ ContentView.Behaviors >

    < ContentView.Triggers >
        < DataTrigger Tipo de objetivo = "ContentView"
            Unión = "{ Unión Fuente = { X : Referencia pedestrianRadio },
                        Camino = IsChecked}"
            Valor = "True">
            < Setter Propiedad = "BackgroundColor" Valor = "{ StaticResource selectedColor }"/>
        </ DataTrigger >
    </ ContentView.Triggers >

    < Imagen Fuente = "{ local : ImageResource Radiolimages .Images.pedestrian.png }"/>
</ ContentView >

< ContentView >
    < ContentView.Behaviors >
        < kit de herramientas : RadioBehavior X : Nombre = "CarRadio" />
    </ ContentView.Behaviors >

    < ContentView.Triggers >
        < DataTrigger Tipo de objetivo = "ContentView"
            Unión = "{ Unión Fuente = { X : Referencia radio de coche },
                        Camino = IsChecked}"
            Valor = "True">
            < Setter Propiedad = "BackgroundColor" Valor = "{ StaticResource selectedColor }"/>
        </ DataTrigger >
    </ ContentView.Triggers >

    < Imagen Fuente = "{ local : ImageResource Radiolimages .Images.car.png }"/>
</ ContentView >

< ContentView >
    < ContentView.Behaviors >
        < kit de herramientas : RadioBehavior X : Nombre = "TrainRadio" />
    </ ContentView.Behaviors >

    < ContentView.Triggers >
        < DataTrigger Tipo de objetivo = "ContentView"
            Unión = "{ Unión Fuente = { X : Referencia trainRadio },
                        Camino = IsChecked}"
            Valor = "True">
            < Setter Propiedad = "BackgroundColor" Valor = "{ StaticResource selectedColor }"/>
        </ DataTrigger >
    </ ContentView.Triggers >

    < Imagen Fuente = "{ local : ImageResource Radiolimages .Images.train.png }"/>
</ ContentView >
```

```

< ContentView >
  < ContentView.Behaviors >
    < kit de herramientas : RadioBehavior X : Nombre = "BusRadio" />
  </ ContentView.Behaviors >

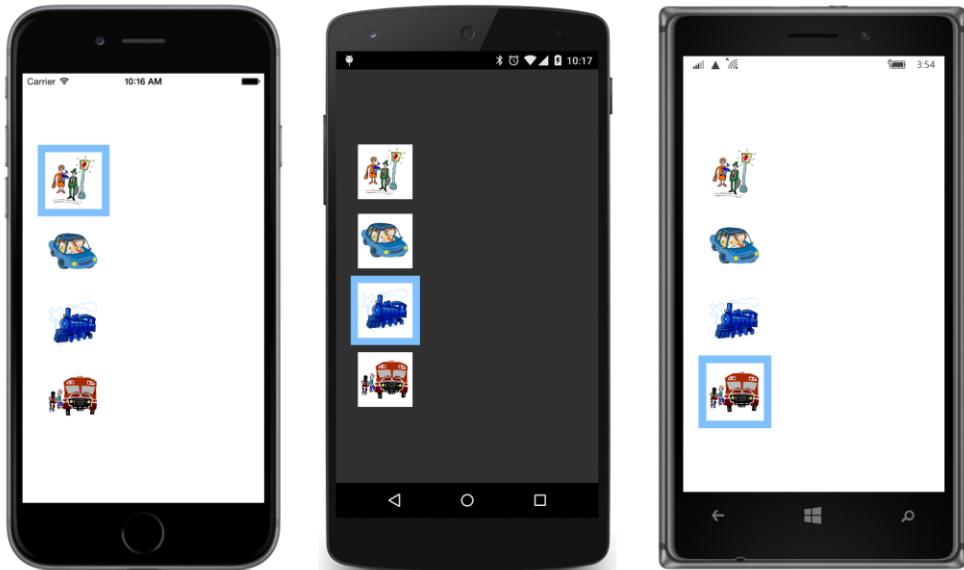
  < ContentView.Triggers >
    < DataTrigger Tipo de objetivo = "ContentView"
      Unión = "{ Unión Fuente = { X : Referencia busRadio },
                  Camino = IsChecked}"
      Valor = "True">
      <Setter Propiedad = "BackgroundColor" Valor = "{ StaticResource selectedColor}" />
    </ DataTrigger >
  </ ContentView.Triggers >

  < Imagen Fuente = "{ local : ImageResource RadiоИmages .Images.bus.png}" />
</ ContentView >
</ StackLayout >
</ Página de contenido >

```

A veces, tendrá que fijar un elemento seleccionado mediante el establecimiento inicial de la Está chequeado propiedad de una de las RadioBehavior a objetos cierto, y otras veces no. Este programa les deja a todos sin control al inicio del programa, pero una vez que el usuario selecciona uno de los artículos, no hay manera de anular la selección de todos ellos.

El factor crucial en este esquema es que la ContentView se le da una significativa Relleno valor por lo que parece rodear la Imagen elemento cuando se selecciona ese elemento:



Por supuesto, incluso con sólo cuatro elementos, el marcado repetitivo se ve un poco de mal agüero. Se podría derivar una clase de ContentView para consolidar el RadioBehavior y DataTrigger interacción, pero que había necesidad de definir una propiedad en esta clase derivada para especificar el mapa de bits particular asociado con el pero-

tonelada, y muy probablemente otra propiedad o un evento para indicar cuando ese artículo ha sido seleccionado. En general, es más fácil mantener el margen de beneficio para cada botón de radio al mínimo mediante la definición de las propiedades comunes usando una Estilo u otros recursos.

Si desea crear efectos visuales más tradicionales de botones de radio, es posible también. El Unicode caracteres \u25CB y \u25C9 se asemejan a la tradicional sin control y se comprueban los círculos y puntos de botón de radio.

los **TraditionalRadios** programa tiene seis botones de radio, pero están divididos en dos grupos de tres botones cada uno, por lo que la Nombre del grupo propiedades se deben establecer para al menos uno de los dos grupos. El programa elige para establecer el Nombre del grupo para *todos* los botones de radio a cualquiera "platformGroup" o "LanguageGroup". Cada RadioBehavior está unido a una horizontal StackLayout que contiene uno

Etiqueta con un DataTrigger que permite cambiar entre el "& #x25CB;" y "& #x25C9;" cuerdas, y un segundo Etiqueta que muestra el texto a la derecha de ese símbolo:

```

< Pagina de contenido xmlns = "http://xamarin.com/schemas/2014/forms" >
    xmlns : X = "Http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns : kit de herramientas =
        "Clr-espacio de nombres: Xamarin.FormsBook.Toolkit; montaje = Xamarin.FormsBook.Toolkit"
    X : Clase = "TraditionalRadios.TraditionalRadiosPage">

    < ContentPage.Resources >
        < ResourceDictionary >
            < X : Cuerda X : Llave = "UncheckedRadio" > & #X25C8; < /X : Cuerda >
            < X : Cuerda X : Llave = "CheckedRadio" > & #X25C9; < /X : Cuerda >
        < / ResourceDictionary >
    < / ContentPage.Resources >

    < Cuadrifila VerticalOptions = "Centro" Relleno = "5, 0" >
        <! - la columna izquierda ->
        < StackLayout Grid.Column = "0" Espaciado = "24" >

            <! - Cabecera ->
            < StackLayout HorizontalOptions = "Inicio" Espaciado = "0" >
                < Etiqueta Texto = "Elegir Plataforma" />
                < BoxView Color = "Accent" HeightRequest = "1" />
            < / StackLayout >

            <! - Pila de botones de radio ->
            < StackLayout Espaciado = "12" >

                < StackLayout Orientación = "Horizontal" >
                    < StackLayout.Behaviors >
                        < kit de herramientas : RadioBehavior X : Nombre = "iosRadio" >
                            Nombre del grupo = "PlatformGroup"
                        < / StackLayout.Behaviors >
                    < / StackLayout Orientación >
                    < Label.Triggers >
                        < DataTrigger Tipo de objetivo = "Etiqueta" >
                            Unión = "{ Unión Fuente = { X : Referencia iosR
                                Camino = Is

```

```
        Valor = "True">
    < Setter Propiedad = "Texto" Valor = "{ StaticResource checkedRadio }"/>
</ DataTrigger >
</ Label.Triggers >
</ Etiqueta >
< Etiqueta Texto = "IOS" />
</ StackLayout >

< StackLayout Orientación = "Horizontal">
    < StackLayout.Behaviors >
        < kit de herramientas : RadioBehavior X : Nombre = "AndroidRadio"
            Nombre del grupo = "PlatformGroup" />
    </ StackLayout.Behaviors >

    < Etiqueta Texto = "{ StaticResource uncheckedRadio }" >
        < Label.Triggers >
            < DataTrigger Tipo de objetivo = "Etiqueta"
                Unión = "{ Unión Fuente = { X : Referencia androidRadio },
                    Camino = isChecked)">
                Valor = "True">
            < Setter Propiedad = "Texto" Valor = "{ StaticResource checkedRadio }"/>
        </ DataTrigger >
        </ Label.Triggers >
    </ Etiqueta >
    < Etiqueta Texto = "Android" />
</ StackLayout >

< StackLayout Orientación = "Horizontal">
    < StackLayout.Behaviors >
        < kit de herramientas : RadioBehavior X : Nombre = "WinPhoneRadio"
            Nombre del grupo = "PlatformGroup" />
    </ StackLayout.Behaviors >

    < Etiqueta Texto = "{ StaticResource uncheckedRadio }" >
        < Label.Triggers >
            < DataTrigger Tipo de objetivo = "Etiqueta"
                Unión = "{ Unión Fuente = { X : Referencia winPhoneRadio },
                    Camino = isChecked)">
                Valor = "True">
            < Setter Propiedad = "Texto" Valor = "{ StaticResource checkedRadio }"/>
        </ DataTrigger >
        </ Label.Triggers >
    </ Etiqueta >
    < Etiqueta Texto = "Windows Phone" />
</ StackLayout >
</ StackLayout >

<! - la columna izquierda ->
< StackLayout Grid.Column = "1" Espaciado = "24" >

    <! - Cabecera ->
    < StackLayout HorizontalOptions = "Inicio" Espaciado = "0" >
        < Etiqueta Texto = "Elija el idioma" />
        < BoxView Color = "Accent" HeightRequest = "1" />
    </ StackLayout >
```

```

</ StackLayout >

<! - Pila de botones de radio ->
< StackLayout Espaciado = "12">

    < StackLayout Orientación = "Horizontal">
        < StackLayout.Behaviors >
            < kit de herramientas : RadioBehavior X : Nombre = "ObjectiveCRadio"
                Nombre del grupo = "LanguageGroup" />
        </ StackLayout.Behaviors >

        < Etiqueta Texto = "{ StaticResource uncheckedRadio }">
            < Label.Triggers >
                < DataTrigger Tipo de objetivo = "Etiqueta"
                    Unión = "{ Unión Fuente = { X : Referencia objectiveCRadio },
                        Camino = IsChecked}">
                    Valor = "True"
                < Setter Propiedad = "Texto" Valor = "{ StaticResource checkedRadio }"/>
            </ DataTrigger >
            </ Label.Triggers >
        </ Etiqueta >
        < Etiqueta Texto = "Objective-C" />
    </ StackLayout >

    < StackLayout Orientación = "Horizontal">
        < StackLayout.Behaviors >
            < kit de herramientas : RadioBehavior X : Nombre = "JavaRadio"
                Nombre del grupo = "LanguageGroup" />
        </ StackLayout.Behaviors >

        < Etiqueta Texto = "{ StaticResource uncheckedRadio }">
            < Label.Triggers >
                < DataTrigger Tipo de objetivo = "Etiqueta"
                    Unión = "{ Unión Fuente = { X : Referencia javaRadio },
                        Camino = IsChecked}">
                    Valor = "True"
                < Setter Propiedad = "Texto" Valor = "{ StaticResource checkedRadio }"/>
            </ DataTrigger >
            </ Label.Triggers >
        </ Etiqueta >
        < Etiqueta Texto = "Java" />
    </ StackLayout >

    < StackLayout Orientación = "Horizontal">
        < StackLayout.Behaviors >
            < kit de herramientas : RadioBehavior X : Nombre = "CSharpRadio"
                Nombre del grupo = "LanguageGroup" />
        </ StackLayout.Behaviors >

        < Etiqueta Texto = "{ StaticResource uncheckedRadio }">
            < Label.Triggers >
                < DataTrigger Tipo de objetivo = "Etiqueta"
                    Unión = "{ Unión Fuente = { X : Referencia cSharpRadio },
                        Camino = IsChecked}">

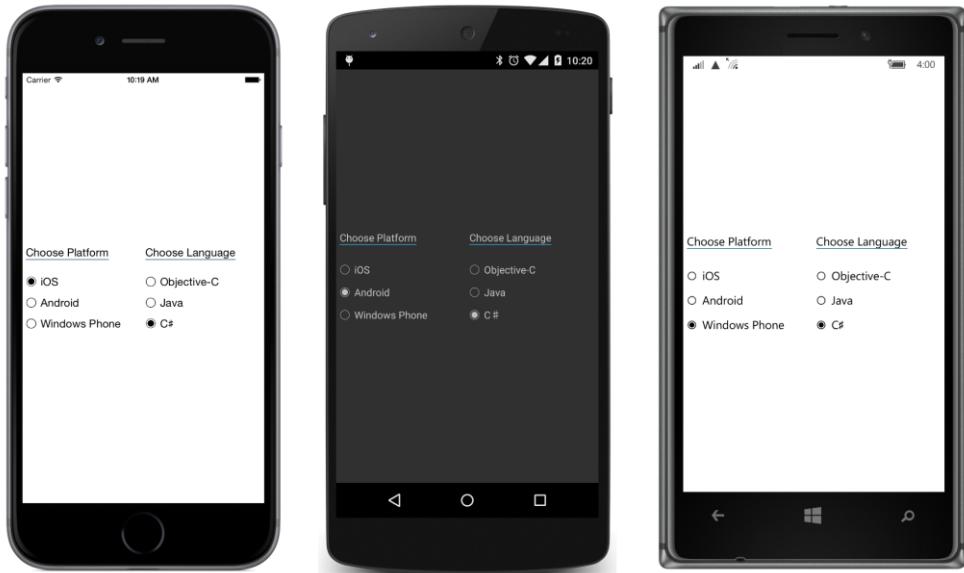
```

```

        Valor = "True">
    <Setter Propiedad = "Texto" Valor = "{ StaticResource checkedRadio }"/>
</DataTrigger>
</Label.Triggers>
</Etiqueta>
<Etiqueta Texto = "C & # x266F;" />
</StackLayout>
</StackLayout>
</Cuadrícula>
</Página de contenido>

```

En el contexto de las interfaces de usuario modernas, estos botones de radio se ven muy pintoresco y pasado de moda, pero al mismo tiempo bastante auténtico:



Fundidos y orientación

Ya en este libro, usted ha visto un par de programas de selección de color que le permiten formar un color de forma interactiva mediante el uso de tres deslizador elementos. La muestra final de este capítulo es otro programa de selección de color, pero éste le da opciones: Contiene tres botones de radio (en realidad, sencillo Etiqueta elementos) etiquetados "RGB Hex", "Float RGB", y "HSL". Estos le permiten seleccionar un color de tres maneras diferentes:

- Como valores hexadecimales de color rojo, verde y azul que van desde 00 a FF.
- Como valores de punto flotante de color rojo, verde y azul que van de 0 a 1.
- Como tonalidad, saturación y luminosidad valores de punto flotante que va de 0 a 1.

Podría parecer a primera vista compleja para cambiar entre estas tres opciones. Se puede imaginar se requiere que el código de redefinir el alcance del deslizador elementos y volver a formatear el texto que se muestra para mostrar los valores. Sin embargo, en realidad se puede definir la interfaz de usuario en XAML.

El primer truco es que el archivo XAML en realidad contiene nueve deslizador elementos con acompañamiento Etiqueta elementos para mostrar los valores. Cada conjunto de tres deslizador y Etiqueta elementos ocupa una StackLayout con su Es visible propiedad ligada a una de las RadioBehavior objetos unidos a los tres botones de radio. El tres StackLayout elementos ocupan una sola célula Cuadrícula, al igual que las fotos de las camisetas en el radiomarcadores y RadioStyle programas.

Pero vamos a hacer que sea más difícil: Al seleccionar uno de los botones de radio, es probable que esperar un conjunto de tres deslizador y Etiqueta elementos para ser reemplazado por otro. En lugar de eso tienen el primer conjunto se desvanecen y el nuevo conjunto de fundido en.

¿Cómo puede hacerse esto?

Vamos a construir la marca. Si sólo quería para reemplazar a uno StackLayout con otra, le obligar a la Es visible propiedad de la StackLayout al Está chequeado propiedad de los correspondientes RadioBehavior:

```
< StackLayout Es visible = "{ Unión Fuente = { X : Referencia hexRadio },
    Camino = IsChecked } ">
```

<! - Trío de Deslizador de etiquetas y elementos ->

```
</ StackLayout >
```

A desvanecerse en lugar de la vieja y se desvanecen en el nuevo, primero se tiene que inicializar el Es visible propiedad de la StackLayout a Falso y adjuntar una DataTrigger que hace referencia al Está chequeado propiedad de la RadioBehavior:

```
< StackLayout Es visible = "False">
    < StackLayout.Triggers >
        < DataTrigger Tipo de objetivo = "StackLayout"
            Unión = "{ Unión Fuente = { X : Referencia hexRadio },
            Camino = IsChecked }"
            Valor = "True" >
            ...
        </ DataTrigger >
    </ StackLayout.Triggers >
```

<! - Trío de Deslizador de etiquetas y elementos ->

```
</ StackLayout >
```

Entonces, en vez de añadir una Setter o dos a la DataTrigger, es necesario agregar una Acción derivado de la EnterActions y ExitActions colecciones:

```
< StackLayout Es visible = "False">
```

```

< StackLayout.Triggers >
    < DataTrigger Tipo de objetivo = "StackLayout"
        Unión = "{ Unión Fuentc = { X : Referencia hexRadio },
                    Camino = IsChecked}"
        Valor = "True">
        < DataTrigger.EnterActions >
            < kit de herramientas : FadeEnableAction Habilitar = "True" />
        </ DataTrigger.EnterActions >

        < DataTrigger.ExitActions >
            < kit de herramientas : FadeEnableAction Habilitar = "False" />
        </ DataTrigger.ExitActions >
    </ DataTrigger >
</ StackLayout.Triggers >

<! - Trío de Deslizador de etiquetas y elementos ->

</ StackLayout >

```

Como se recordará, el EnterActions se invoca cuando la condición sea verdadera (que en este caso es cuando el Está chequeado propiedad de los correspondientes RadioBehavior es Ciento), y el ExitActions se invoca cuando la condición se convierte en falsa.

esta hipotética FadeEnableAction clase tiene una propiedad booleana llamada Habilitar. Cuando el ENAble propiedad es Ciento, queremos FadeEnableAction usar el Desvanecerse hacia método de extensión para animar la Opacidad propiedad de 0 (invisible) a 1 (totalmente visible). Cuando Habilitar es Falso, queremos Desvanecerse hacia para animar el Opacidad de 1 a 0. Tenga en cuenta que a medida que uno StackLayout (y sus hijos) desvanece, otro se desvanece al mismo tiempo en.

sin embargo, el StackLayout no serán visibles en absoluto a menos FadeEnableAction comienza estableciendo EsVisible a cierto cuando Habilitar se establece en Ciento. Del mismo modo, cuando Habilitar se establece en Falso, FadeEnableAction debe concluir estableciendo Es visible de regreso falso.

Durante la transición entre dos conjuntos de deslizador y Etiqueta elementos, es probable que no quieren que los dos conjuntos que respondan a la entrada del usuario. Por esta razón, FadeEnableAction También debe manipular el EsActivado propiedad de la StackLayout, que activa o desactiva todos sus hijos. Desde dos animaciones van a ir en forma simultánea, como uno StackLayout desvanece y los demás se desvanece en-que tiene sentido para cambiar la Está habilitado a mitad de camino a través de la propiedad de animación.

Aquí es una FadeEnableAction clase de **Xamarin.FormsBook.Toolkit** que satisface todos estos criterios:

```

espacio de nombres Xamarin.FormsBook.Toolkit
{
    clase pública FadeEnableAction : TriggerAction < VisualElement >
    {
        público FadeEnableAction ()
        {
            Longitud = 500;
        }

        public bool Habilitar { conjunto ; obtener ; }
    }
}

```

Vamos a darnos un nuevo desafío. En el capítulo 17, “El dominio de la red”, en la sección “En respuesta a los cambios de orientación”, que vio cómo utilizar el Cuadrícula a cambiar el diseño entre los modos vertical y horizontal. Básicamente, todo el diseño de la página se divide aproximadamente por la mitad, y se convierte en dos niños de una Cuadrícula. En el modo vertical, esos dos niños van en dos filas de la Cuadrícula, y en modo horizontal, van en dos columnas.

Puede ser algo como esto manejado por un comportamiento? Con capacidad para una respuesta generalizada a la orientación sería difícil, pero un enfoque simple podría ser la de asumir que en el modo vertical, la segunda fila se debe autosized mientras que la primera fila usa el resto del espacio disponible. En el modo horizontal, la pantalla simplemente se divide en partes iguales a la mitad. Así es como el **GridRgbSliders** programa en el Capítulo 17 trabajó, y también el **MandelbrotXF** programa en el capítulo 20.

El seguimiento GridOrientationBehavior se puede conectar solamente a una Cuadricula. los Cuadricula debe no disponen de definiciones fila o definiciones de columna definido el comportamiento se encarga de eso-y debe contener sólo dos hijos. El comportamiento supervisa la SizeChanged caso de la Cuadricula. Cuando que los cambios de tamaño, la Comportamiento establece las definiciones de filas y columnas de la Cuadricula y las filas y columnas configuración de los dos hijos de la Cuadricula:

```
espacio de nombres Xamarin.FormsBook.Toolkit  
{  
    // Asume cuadrícula con dos niños sin  
    //         de fila o columna definiciones establecidas.  
    clase pública GridOrientationBehavior : Comportamiento < Cuadrícula >  
}
```

```
protegido override void OnAttachedTo ( Cuadrícula cuadrícula)
{
    base .OnAttachedTo (rejilla);

    // Añadir fila y columna definiciones.
    grid.RowDefinitions.Add ( nuevo RowDefinition ());
    grid.RowDefinitions.Add ( nuevo RowDefinition ());
    grid.ColumnDefinitions.Add ( nuevo ColumnDefinition ());
    grid.ColumnDefinitions.Add ( nuevo ColumnDefinition ());

    grid.SizeChanged += OnGridSizeChanged;
}

protegido override void OnDetachingFrom ( Cuadrícula cuadrícula)
{
    base .OnDetachingFrom (rejilla);
    grid.SizeChanged -= OnGridSizeChanged;
}

private void OnGridSizeChanged ( objeto remitente, EventArgs args)
{
    Cuadrícula rejilla = ( Cuadrícula )remitente;

    Si (Grid.Width <= 0 || grid.Height <= 0)
        regreso ;

    // Modo retrato
    Si (Grid.Height> grid.Width)
    {
        // Establecer las definiciones de fila.
        grid.RowDefinitions [0] = .height nuevo GridLength (1, GridUnitType .Estrella);
        grid.RowDefinitions [1].height = GridLength .Auto;

        // Establecer definiciones de columna.
        grid.ColumnDefinitions [0] = .Width nuevo GridLength (1, GridUnitType .Estrella);
        grid.ColumnDefinitions [1].Width = nuevo GridLength (0);

        // Posición primer hijo.
        Cuadrícula .SetRow (grid.Children [0], 0);
        Cuadrícula .SetColumn (grid.Children [0], 0);

        // Posición segundo hijo.
        Cuadrícula .SetRow (grid.Children [1], 1);
        Cuadrícula .SetColumn (grid.Children [1], 0);
    }

    // Modo paisaje
    más
    {
        // Establecer las definiciones de fila.
        grid.RowDefinitions [0] = .height nuevo GridLength (1, GridUnitType .Estrella);
        grid.RowDefinitions [1].height = nuevo GridLength (0);

        // Establecer definiciones de columna.
        grid.ColumnDefinitions [0] = .Width nuevo GridLength (1, GridUnitType .Estrella);
        grid.ColumnDefinitions [1].Width = nuevo GridLength (1, GridUnitType .Estrella);
    }
}
```

```
// Posición primer hijo.  
Cuadrícula .SetRow (grid.Children [0], 0);  
Cuadrícula .SetColumn (grid.Children [0], 0);  
  
// Posición segundo hijo.  
Cuadrícula .SetRow (grid.Children [1], 0);  
Cuadrícula .SetColumn (grid.Children [1], 1);  
}  
}  
}
```

Ahora vamos a poner todo junto en una llamada al programa **MultiColorSliders**. La columna vertebral del programa es el **ColorViewModel** presentado en el capítulo 18, "MVVM," y se puede encontrar en el **Xamarin.FormsBook.Toolkit** biblioteca. Una instancia de **ColorViewModel** se establece como el **BindingContext** del Cuadricula que incluye todo el contenido de la página. Los tres conjuntos de deslizador y Etiqueta elementos contienen todos los enlaces a la Rojo, verde, azul, tonalidad, Saturación, y Luminosidad propiedades de ese modelo de vista. Para la opción hexadecimal, la **DoubleToIntConverter** introducido en el capítulo 17 conversos de la doble

valores de la Rojo verde, y Azul propiedades de los números enteros con una multiplicación por 255 para la visualización por cada uno Etiqueta.

Aquí está el archivo XAML. Es bastante largo, ya que contiene tres grupos de tres deslizador y Etiqueta elementos, pero varios comentarios ayudan a quiarlo a través de las distintas secciones:

```
< Pagina de contenido xmlns = "http://xamarin.com/schemas/2014/forms"
    xmlns : X = "http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns : kit de herramientas =
        "clr-espacio de nombres: Xamarin.FormsBook.Toolkit; montaje = Xamarin.FormsBook.Toolkit"
    X : Clase = "MultiColorSliders.MultiColorSlidersPage">

< ContentPage.Padding >
    < OnPlatform X : TypeArguments = "Espesor"
        iOS = "0, 20, 0, 0" />
</ ContentPage.Padding >

< ContentPage.Resources >
    < ResourceDictionary >
        < kit de herramientas : ColorViewModel X : Llave = "ColorViewModel" />
        < kit de herramientas : DoubleToIntConverter X : Llave = "DoubleToInt" />
        < Estilo X : Llave = "BaseStyle" Tipo de objetivo = "Etiqueta">
            < Setter Propiedad = "HorizontalTextAlignment" Valor = "Center" />
        </ Estilo >
        < Estilo X : Llave = "UnselectedStyle" Tipo de objetivo = "Etiqueta"
            Residencia en = "( StaticResource baseStyle ) ">
            < Setter Propiedad = "TextColor" Valor = "Gris" />
        </ Estilo >
        < Estilo X : Llave = "SelectedStyle" Tipo de objetivo = "Etiqueta"
            Residencia en = "( StaticResource baseStyle ) ">
```

```

< Setter Propiedad = "TextColor" Valor = "Acento" />
< Setter Propiedad = "Escala" Valor = "1.5" />
</ Estilo >

<! - estilo implícito para las etiquetas debajo de los deslizadores ->
< Estilo Tipo de objetivo = "Etiqueta" Residencia en = "[ StaticResource baseStyle ]">
</ ResourceDictionary >
</ ContentPage.Resources >

< Cuadricula >
< Grid.BindingContext >
    < kit de herramientas : ColorViewModel Alfa = "1" />
</ Grid.BindingContext >

<! - El GridOrientationBehavior se encarga de la fila y
definiciones de columna y la configuración de fila y columna
de los dos hijos de cuadrícula. ->
< Grid.Behaviors >
    < kit de herramientas : GridOrientationBehavior />
</ Grid.Behaviors >

<! - En primer hijo de la cuadrícula es en la parte superior o en la izquierda. ->
< BoxView Color = "{ Unión Color }" />

<! - El segundo niño de la red es en la parte inferior o en la derecha. ->
< StackLayout Relleno = "10" >

<! - cuadrícula de tres columnas para las etiquetas de radio ->
< Cuadricula >
    < Etiqueta Texto = "RGB Hex" Grid.Column = "0"
        Estilo = "{ StaticResource unselectedStyle }" >
        < Label.Behaviors >
            < kit de herramientas : RadioBehavior X : Nombre = "HexRadio"
                Está chequeado = "True" />
        </ Label.Behaviors >

        < Label.Triggers >
            < DataTrigger Tipo de objetivo = "Etiqueta"
                Unión = "{ Unión Fuente = { X : Referencia hexRadio },
                Camino = IsChecked }"
                Valor = "True" >
                < Setter Propiedad = "Estilo" Valor = "[ StaticResource selectedStyle ]" />
            </ DataTrigger >
        </ Label.Triggers >
    </ Etiqueta >

    < Etiqueta Texto = "Float RGB" Grid.Column = "1"
        Estilo = "{ StaticResource unselectedStyle }" >
        < Label.Behaviors >
            < kit de herramientas : RadioBehavior X : Nombre = "FloatRadio" />
        </ Label.Behaviors >

        < Label.Triggers >
            < DataTrigger Tipo de objetivo = "Etiqueta"

```

```

        Unión = "{ Unión Fuente = { X : Referencia floatRadio },
                    Camino = isChecked}"*
        Valor = "True">
    < Setter Propiedad = "Estilo" Valor = "[ StaticResource selectedStyle ]"/>
</ DataTrigger >
</ Label.Triggers >
</ Etiqueta >

< Etiqueta Texto = "HSL" Grid.Column = "2"*
    Estilo = "[ StaticResource unselectedStyle ]">
< Label.Behaviors >
    < kit de herramientas : RadioBehavior X : Nombre = "HslRadio" />
</ Label.Behaviors >

< Label.Triggers >
    < DataTrigger Tipo de objetivo = "Etiqueta"*
        Unión = "{ Unión Fuente = { X : Referencia hslRadio },
                    Camino = isChecked}"*
        Valor = "True">
    < Setter Propiedad = "Estilo" Valor = "[ StaticResource selectedStyle ]"/>
</ DataTrigger >
</ Label.Triggers >
</ Etiqueta >
</ Cuadrícula >

<! - cuadrícula de celda única de tres conjuntos de deslizadores y etiquetas ->
< Cuadrícula >

<! - StackLayout de deslizadores y etiquetas RGB Hex ->
< StackLayout >
    < StackLayout.Triggers >
        < DataTrigger Tipo de objetivo = "StackLayout"*
            Unión = "{ Unión Fuente = { X : Referencia hexRadio },
                        Camino = isChecked}"*
            Valor = "True">
            < DataTrigger.EnterActions >
                < kit de herramientas : FadeEnableAction Habilitar = "True" />
            </ DataTrigger.EnterActions >

            < DataTrigger.ExitActions >
                < kit de herramientas : FadeEnableAction Habilitar = "False" />
            </ DataTrigger.ExitActions >
        </ DataTrigger >
    </ StackLayout.Triggers >

    < deslizador Valor = "[ Unión rojo , Modo = TwoWay ]"/>

    < Etiqueta Texto = "[ Unión rojo , StringFormat = 'Red = {0: X2}',*
                    Convertidor = [ StaticResource doubleToInt ],
                    ConverterParameter = 255 ]"/>

    < deslizador Valor = "[ Unión Verde , Modo = TwoWay ]"/>

    < Etiqueta Texto = "[ Unión Verde , StringFormat = 'Verde = {0: X2}',*
                    Convertidor = [ StaticResource doubleToInt ],
                    ConverterParameter = 255 ]"/>

```

```

        Convertidor = { StaticResource doubleToInt },
        ConverterParameter = 255 }"/>

    < deslizador Valor = "{ Unión Azul , Modo = TwoWay}" />

    < Etiqueta Texto = "{ Unión Azul , StringFormat = 'Azul = {0: X2}',

        Convertidor = { StaticResource doubleToInt },
        ConverterParameter = 255 }"/>

</ StackLayout >

<! - StackLayout de deslizadores y etiquetas flotador RGB ->
< StackLayout Es visible = "False">
    < StackLayout.Triggers >
        < DataTrigger Tipo de objetivo = "StackLayout"
            Unión = "{ Unión Fuente = { X : Referencia floatRadio },
                Camino = IsChecked}"
            Valor = "True">
            < DataTrigger.EnterActions >
                < kit de herramientas : FadeEnableAction Habilitar = "True" />
            </ DataTrigger.EnterActions >

            < DataTrigger.ExitActions >
                < kit de herramientas : FadeEnableAction Habilitar = "False" />
            </ DataTrigger.ExitActions >
        </ DataTrigger >
        < StackLayout.Triggers >

        < deslizador Valor = "{ Unión rojo , Modo = TwoWay}" />
        < Etiqueta Texto = "{ Unión rojo , StringFormat = Rojo = {0: F2}}"/>
        < deslizador Valor = "{ Unión Verde , Modo = TwoWay}" />
        < Etiqueta Texto = "{ Unión Verde , StringFormat = Verde = {0: F2}}"/>
        < deslizador Valor = "{ Unión Azul , Modo = TwoWay}" />
        < Etiqueta Texto = "{ Unión Azul , StringFormat = 'Azul = {0: F2}' }"/>
    </ StackLayout >

    <! - StackLayout para los reguladores HSL y etiquetas ->
    < StackLayout Es visible = "False">
        < StackLayout.Triggers >
            < DataTrigger Tipo de objetivo = "StackLayout"
                Unión = "{ Unión Fuente = { X : Referencia hslRadio },
                    Camino = IsChecked}"
                Valor = "True">
                < DataTrigger.EnterActions >
                    < kit de herramientas : FadeEnableAction Habilitar = "True" />
                </ DataTrigger.EnterActions >

                < DataTrigger.ExitActions >
                    < kit de herramientas : FadeEnableAction Habilitar = "False" />
                </ DataTrigger.ExitActions >
            </ DataTrigger >
        < StackLayout.Triggers >

        <! - Trío de Deslizador de etiquetas y elementos ->

```

```

< deslizador Valor = "{ Unión Matiz , Modo = TwoWay}">
< Etiqueta Texto = "{ Unión Matiz , StringFormat = 'Hue = {0: F2}'}">
< deslizador Valor = "{ Unión Saturación , Modo = TwoWay}">
< Etiqueta Texto = "{ Unión Saturación , StringFormat = 'Saturation = {0: F2}'}">
< deslizador Valor = "{ Unión Luminosidad , Modo = TwoWay}">
< Etiqueta Texto = "{ Unión Luminosidad , StringFormat = 'Luminosidad = {0: F2}'}">

</ StackLayout >
</ Cuadrícula >
</ StackLayout >
</ Cuadrícula >
</ Página de contenido >

```

Se puede recordar que la **ColorViewModel** clase introducido en el capítulo 18 redondea los componentes de color, tanto que entra y que sale del modelo de vista. **MultiColorSliders** pasa a ser el programa que revela un problema con los valores no redondeados. Aquí está el problema:

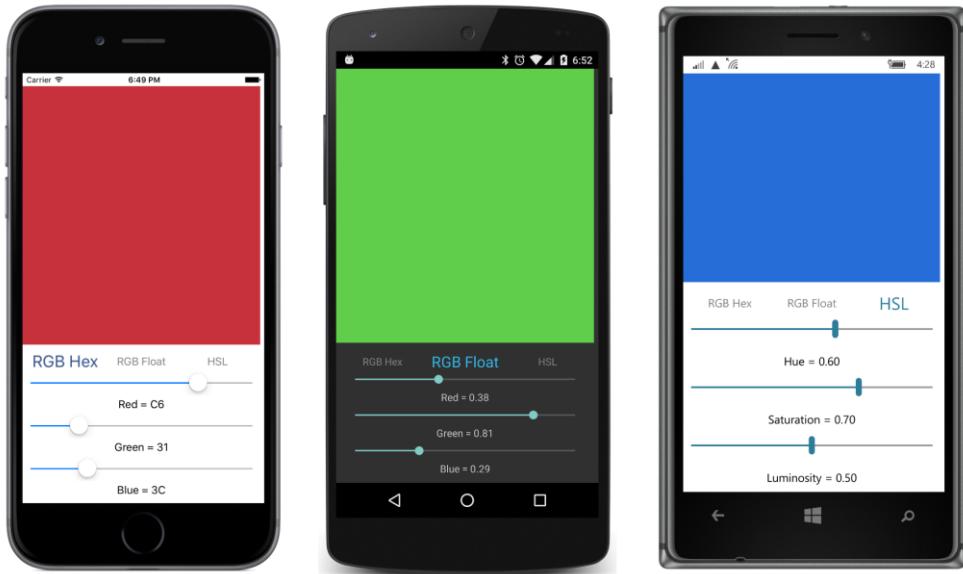
Para Android, **Xamarin.Forms** implementa la deslizador usando un Barra de búsqueda, y el Android Barra de búsqueda sólo tiene número entero Progreso valores que van desde 0 hasta el número entero **Max** propiedad. Para convertir a la de punto flotante **Valor** propiedad de la deslizador, **Xamarin.Forms** establece el **Max** propiedad de la Barra de búsqueda a 1000 y a continuación, realiza un cálculo basado en el **Mínimo** y **Máximo** propiedades de la Deslizador.

Esto significa que cuando **Mínimo** y **Máximo** tener sus valores por defecto de 0 y 1, respectivamente, la **Valor** propiedad sólo aumenta en incrementos de 0,001, y siempre es representable con tres posiciones decimales.

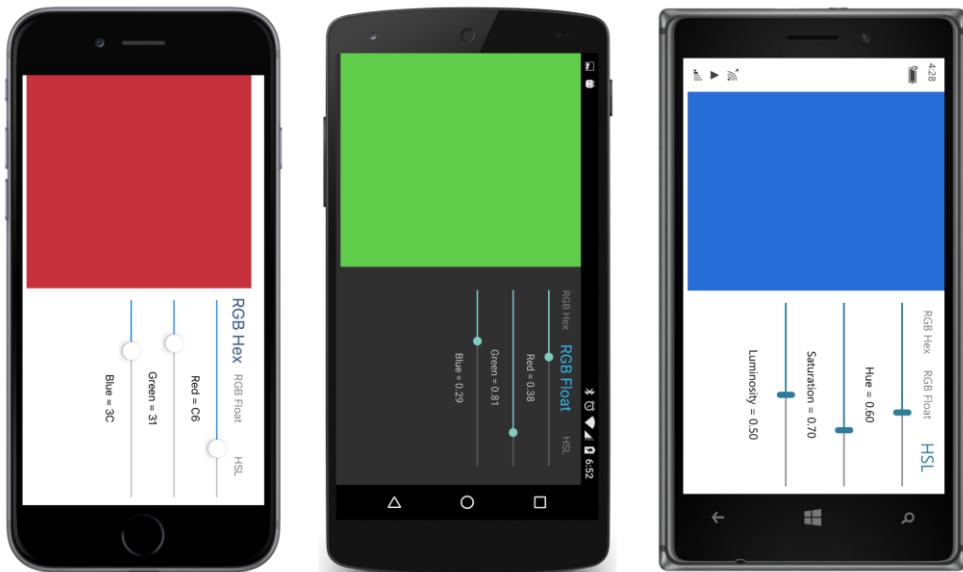
sin embargo, el **ColorViewModel** utiliza el **Color** estructura para convertir entre representaciones RGB y HSL, y en este programa en particular todas las propiedades que representan valores RGB y HSL están obligados a deslizador elementos. Incluso si los valores de la Rojo, verde, y Azul propiedades establecidas por el deslizador elementos se redondean al 0.001 más cercano, la resultante La saturación de color, y Luminosidad Los valores tendrán más de tres cifras decimales. Si estos valores no son redondeados por el modelo de vista, eso es un problema. Cuando el **Valor** propiedades de la deslizador elementos se establecen a partir de estos valores, la deslizador efectivamente se redondea hasta tres cifras decimales y luego desencadena una **PropertyChanged** caso de que el **ColorViewModel** responde a la creación de una nueva **Color**, lo que se traduce en nuevas Rojo verde, y Azul propiedades, y un bucle infinito sobreviene.

La solución, como se vio en el capítulo 18 fue la de añadir al redondeo **ColorViewModel**. Eso evita el bucle infinito.

Aquí está el programa que se ejecuta en modo vertical. Cada plataforma muestra una opción diferente seleccionado, pero usted tendrá que ejecutar el programa usted mismo para ver la animación de la decoloración:



Convertir este libro (o la pantalla del ordenador o tal vez su cabeza) hacia los lados, y verá cómo el programa responde a modo de paisaje:



Tal vez la mejor parte de la **MultiColorSliders** programa es el archivo de código subyacente, que contiene más que una llamada a `InitializeComponent`:

```
espacio de nombres MultiColorSliders
```

```
{
```

```
público clase parcial MultiColorSlidersPage : Pagina de contenido
{
    público MultiColorSlidersPage ()
    {
        InitializeComponent ();
    }
}
```

Hay, por supuesto, una cantidad considerable de código en el apoyo **MultiColorSliders**, que consiste en dos **Comportamiento <T> derivados**, una **Acción <T> derivado**, una **IValueConverter implementación**, y una **INotifyPropertyChanged aplicación** que funciona como un modelo de vista.

Sin embargo, todo este código se aísla en componentes reutilizables, lo que hace que este programa sea un modelo de la filosofía de diseño MVVM.

capítulo 24

navegación de la página

Los diferentes tipos de entornos informáticos tienden a desarrollar diferentes metáforas para presentar la información al usuario. A veces una metáfora desarrollada dentro de un medio ambiente es tan bueno que influye en otros entornos.

Tal es el caso de la página y la navegación metáfora que se desarrolló en la World Wide Web. Antes de eso, las aplicaciones informáticas de escritorio simplemente no estaban organizadas en torno al concepto de páginas navegables. Pero la web demostró la potencia y la comodidad de la página de la metáfora, y ahora los sistemas operativos móviles y de escritorio en general, apoyar una arquitectura basada en la página, y muchas aplicaciones se han aprovechado de eso.

Una arquitectura de la página es particularmente popular en las aplicaciones móviles, y por eso este tipo de arquitectura se apoya en **Xamarin.Forms**. Una aplicación Xamarin.Forms puede contener varias clases que se derivan de **Página de contenido**, y el usuario puede navegar entre estas páginas. (En el siguiente capítulo, verá varias alternativas a **Página de contenido**. Esos otros tipos de páginas también pueden participar en la navegación.)

En general, una página incluirá una botón (o tal vez una Etiqueta o un Imagen con un TapGestureRecognizer) que el usuario toca a desplazarse a otra página. A veces, esa segunda página le permitirá obtener una navegación a otras páginas.

Pero también debe haber una forma para que el usuario para volver a la página anterior, y aquí es donde las diferencias de plataforma comenzar a manifestar a sí mismos: los dispositivos Android y Windows Phone incorporan una norma **Espalda** botón (simbolizado como una flecha hacia la izquierda o triángulo) en la parte inferior de la pantalla; dispositivos IOS no, y tampoco lo hace Windows que se ejecuta en el escritorio o una tableta.

Además, como se verá, el software estándar **Espalda** Existen botones en la parte superior de algunas páginas navegables (pero no todos) como parte de la interfaz de usuario estándar de iOS y Android, y también por el tiempo de ejecución de Windows cuando se ejecuta en computadoras de escritorio o tabletas.

Desde la perspectiva del programador, navegación de páginas se implementa con el concepto familiar de una pila. Cuando navega por una página a otra, la nueva página se inserta en la pila y se convierte en la página activa. Cuando la segunda página vuelve de nuevo a la primera página, una página se extrae de la pila, y la nueva página superior se convierte en activo. La aplicación tiene acceso a la pila de navegación que mantiene Xamarin.Forms para la aplicación y es compatible con los métodos para manipular la pila mediante la inserción de páginas o la eliminación de ellos.

Una aplicación que se estructura en torno a varias páginas siempre tiene una página que es especial porque es el punto de inicio de la aplicación. Esto a menudo se llama el **principal** página, o la **casa** página, o la **comienzo** página.

Todas las demás páginas de la aplicación son intrínsecamente diferente de la página de inicio, sin embargo, debido a que se dividen en dos categorías diferentes: Páginas modales y no modales páginas.

páginas modales y no modales páginas

En el diseño de interfaz de usuario, “modal” se refiere a algo que requiere interacción con el usuario antes de la aplicación puede continuar. Las aplicaciones informáticas en el escritorio veces muestran ventanas modales o cuadros de diálogo modales. Cuando se muestra una de estas modales objetos, el usuario no puede simplemente utilizar el ratón para cambiar a la ventana principal de la aplicación. El objeto modal exige más atención por parte del usuario antes de que desaparezca.

Una ventana o diálogo que no es modal a menudo se llama *modal* cuando es necesario distinguir entre los dos tipos.

El sistema de navegación-Xamarin.Forms página asimismo implementa páginas modales y no modales mediante la definición de dos métodos distintos que una página pueda llamar para navegar a otra página:

Tarea PushAsync (Página)

Tarea PushModalAsync (Página)

La página para navegar hasta se pasa como argumento. Como el nombre del segundo método implica, que navega a una página modal. **Lo simple PushAsync Método navega a una página no modal, que en la programación de la vida real es el tipo de página más común.**

Otros dos métodos se definen para volver a la página anterior:

Tarea <Página> PopAsync ()

Tarea <Página> PopModalAsync ()

En muchos casos, una aplicación no necesita llamar PopAsync directamente si se basa en la navegación de regreso proporcionada por el sistema telefónico o de funcionamiento.

los Tarea valor de retorno y la asíncrono sufijo de éstos empujar y Popular nombres de los métodos que se indican son asíncronas, pero esto no quiere decir que una página navegado ejecuta en un hilo diferente de la ejecución! Lo que la finalización de la tarea Indica que se ha discutido más adelante en este capítulo.

Estos cuatro métodos-así como otros métodos de navegación y propiedades-se definen en el INavigation interfaz. El objeto que implementa esta interfaz es interna a Xamarin.Forms, pero VisualElement define una propiedad de sólo lectura denominado Navegación de tipo INavigation, y esto le da acceso a los métodos de navegación y propiedades.

Esto significa que puede utilizar estos métodos de navegación desde una instancia de cualquier clase que deriva de VisualElement. En general, sin embargo, va a utilizar el Navegación propiedad del objeto de la página, por lo que el código para navegar a una nueva página a menudo se parece a esto:

```
esperar Navigation.PushAsync ( nuevo MyNewPage () );
```

o esto:

```
esperar Navigation.PushModalAsync ( nuevo MyNewModalPage () );
```

La diferencia entre las páginas modales y no modales implica sobre todo la interfaz de usuario que el sistema operativo proporciona en la página para volver a la página anterior. Esta diferencia varía según la plataforma. Una mayor diferencia en la interfaz de usuario entre las páginas no modales y modales existe en IOS y el escritorio de Windows o comprimidos; algo menor diferencia se encuentra en Android y las plataformas de teléfonos Windows.

Por lo general, va a utilizar páginas modales cuando su aplicación necesita alguna información del usuario y no desea que el usuario pueda volver a la página anterior hasta que se proporcione esa información. Para trabajar en todas las plataformas, una página modal debe proporcionar su propia interfaz de usuario para navegar de nuevo a la página anterior.

Vamos a comenzar por la exploración de la diferencia entre las páginas no modales y modales con más detalle. los **ModelessAndModal** programa contiene tres páginas con los nombres de las clases MainPage, ModalPage, y ModelessPage. Las páginas en sí son bastante simple, así que para mantener el grueso archivo a un mínimo, se trata de páginas de código solamente. En una aplicación real, las páginas pueden ser implementados con XAML o en el otro extremo-generado dinámicamente por el código. (Usted verá ejemplos de las dos opciones más adelante en este capítulo).

Página principal crea dos Botón elementos, uno que navega a una página no modal y el otro que navega a una página modal. Observe la Título conjunto de propiedades en la parte superior del constructor. Esta Título propiedad no tiene efecto en una aplicación de una sola página, pero juega un papel importante en las aplicaciones de varias páginas:

```
público clase Página principal : Página de contenido
{
    público Página principal()
    {
        title = "Página principal";

        Botón gotoModelessButton = nuevo Botón
        {
            text = "Ir a la Página no modal",
            HorizontalOptions = LayoutOptions.Centrar,
            VerticalOptions = LayoutOptions.CenterAndExpand
        };
        gotoModelessButton.Clicked += asíncrono (Remitente, args) =>
        {
            esperar Navigation.PushAsync ( nuevo ModelessPage () );
        };

        Botón gotoModalButton = nuevo Botón
        {
            text = "Ir a la Página modal",
            HorizontalOptions = LayoutOptions.Centrar,
            VerticalOptions = LayoutOptions.CenterAndExpand
        };
    }
}
```

```
gotoModalButton.Clicked += asincrono (Remitente, args) =>
{
    esperar Navigation.PushModalAsync ( nuevo ModalPage ());
};

content = nuevo StackLayout
{
    Los niños =
    {
        gotoModelessButton,
        gotoModalButton
    }
};
}
```

los hecho clic controlador para el primer Botón llamadas PushAsync con una nueva instancia de ModelessPage, y las segundas llamadas PushModalAsync con una nueva instancia de ModalPage. los hecho clic manipuladores se señalan con el asincrono palabra clave y llamar a la empujar con métodos esperar.

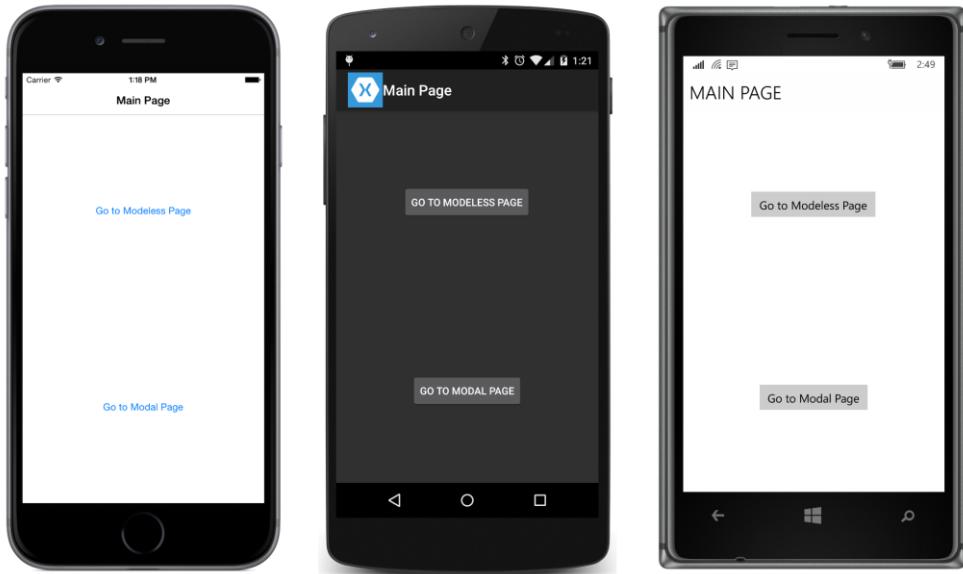
Un programa que realiza llamadas a PushAsync o PushModalAsync debe tener un poco diferente código de inicio en el constructor de la Aplicación clase. En lugar de establecer la Pagina principal propiedad de Aplicación a una instancia de página entera de la aplicación, una instancia de página de inicio de la aplicación generalmente se pasa a la NavigationPage constructor, y esto se establece en el Pagina principal propiedad.

He aquí cómo el constructor de su Aplicación clase normalmente se ve cuando la aplicación incorpora la navegación de páginas:

```
clase pública Aplicación : Solicitud
{
    público App ()
    {
        MainPage = nuevo NavigationPage ( nuevo Pagina principal ());
    }
    ...
}
```

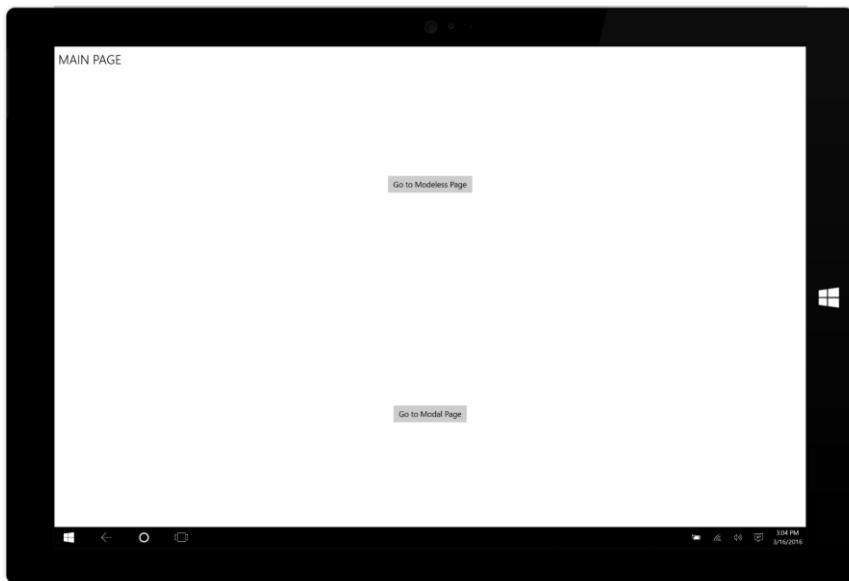
La mayoría de Aplicación las clases en todos los programas de este capítulo contienen código similar. Como alternativa, se puede crear una instancia NavigationPage mediante el uso de su constructor sin parámetros y luego llamar al PushAsync método de la NavigationPage para ir a la página principal.

El uso de NavigationPage da como resultado una diferencia visible en la página. los Título propiedad se muestra en la parte superior de Pagina principal, y está acompañado por el icono de la aplicación en la pantalla de Android:

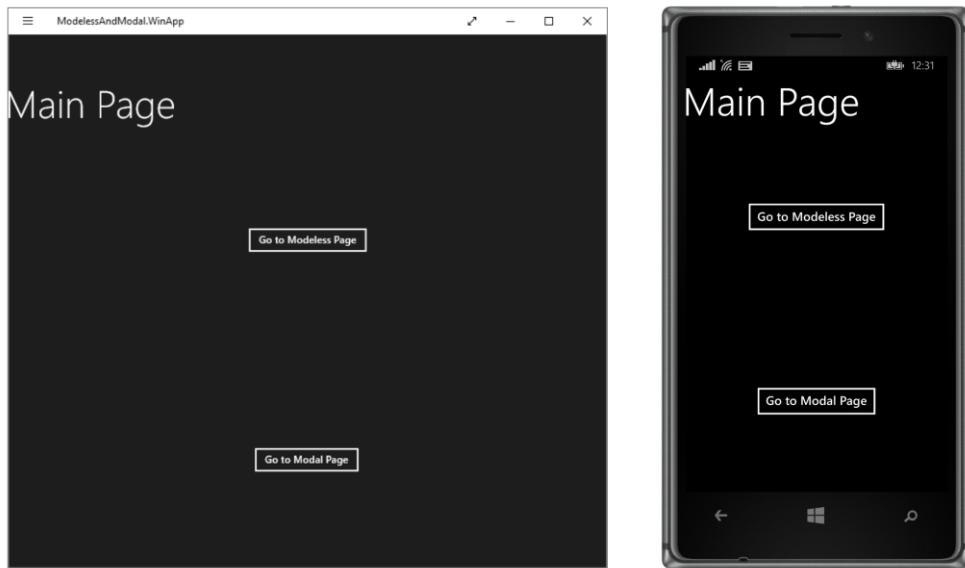


Otra gran diferencia es que ya no es necesario establecer Relleno en la página iOS para evitar sobrescribir la barra de estado en la parte superior de la pantalla.

El título también se muestra en la parte superior del programa de Windows 10 que se ejecuta en el modo de tableta:



Un título bastante más grande se muestra en el teléfono 8.1 plataformas de Windows 8.1 y Windows:



al hacer clic en el **Ir a la Página no modal** botón hace que el siguiente código para ejecutar:

```
esperar Navigation.PushAsync ( nuevo ModelessPage () );
```

Este código crea un nuevo ModelessPage y navega a esa página.

los ModelessPage clase define una Título propiedad con el texto "no modal página" y una Botón

El elemento marcado como **Volver al inicio de** con un hecho clic controlador que llama PopAsync:

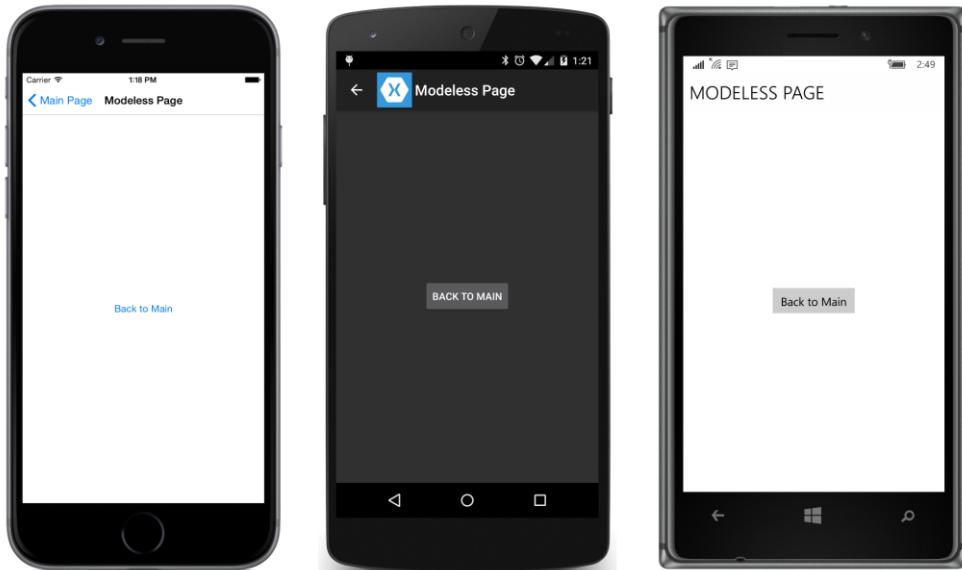
```
clase pública ModelessPage : Pagina de contenido
{
    público ModelessPage ()
    {
        title = "Modal Página";

        Botón goBackButton = nuevo Botón
        {
            text = "Volver al inicio de",
            HorizontalOptions = LayoutOptions .Centrar,
            VerticalOptions = LayoutOptions .Centrar
        };
        goBackButton.Clicked += asíncrono (Remitente, args) =>
        {
            esperar Navigation.PopAsync ();
        };

        Content = goBackButton;
    }
}
```

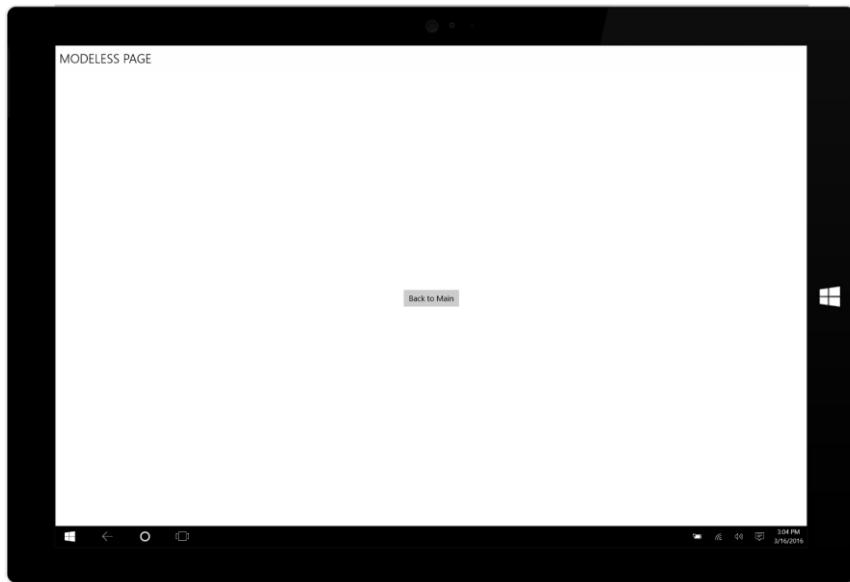
En realidad no necesita el **Volver al inicio de** botón en las páginas de iOS y Android porque uno de izquierda

que apunta la flecha en la parte superior de la página que realiza la misma función. El teléfono de Windows que no necesita Botón ya sea porque tiene una **Espalda** botón en la parte inferior de la pantalla, al igual que el dispositivo Android:

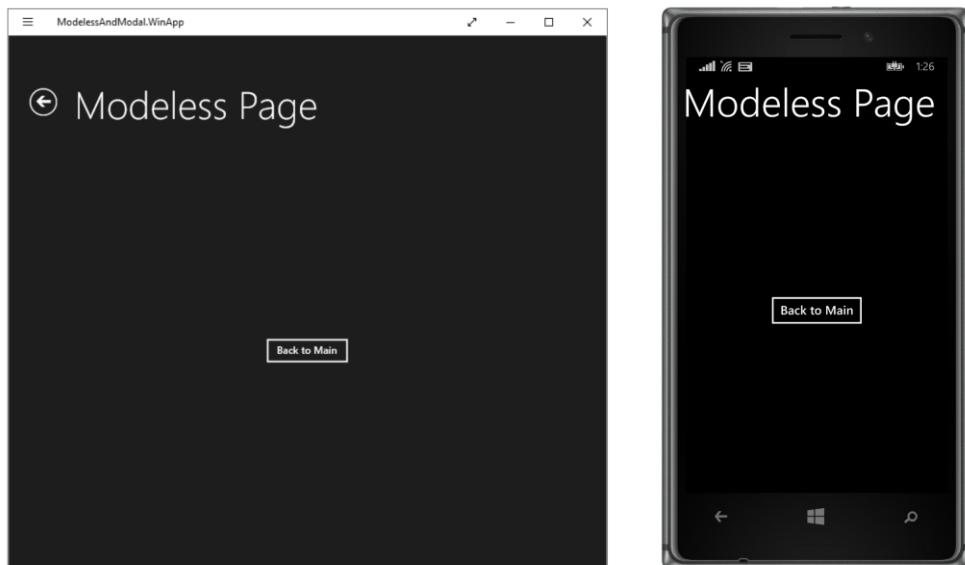


Esa zona superior de las páginas de iOS y Android se llama *barra de navegación*. En esa barra de navegación, tanto en el iOS y Android páginas muestran la Título propiedad de la página actual, y la página también muestra el IOS Título propiedad de la página anterior en otro color.

Un programa que se ejecuta en el modo de tableta bajo Windows 10 contiene una **Espalda** botón en la esquina inferior izquierda, justo a la derecha del logotipo de Windows:



Por el contrario, el programa de Windows 8.1 muestra un botón en forma de flecha de un círculo para navegar de vuelta a la página anterior. La pantalla de Windows Phone 8.1 no necesita ese botón, ya que tiene una **Espalda** botón en la parte inferior de la pantalla:



En resumen, no es necesario incluir su propia **Volver al inicio de** botón (o su equivalente) en una página no modal. O bien la interfaz de navegación o el propio dispositivo proporciona una **Espalda** botón.

Volvamos a Página principal. Al hacer clic en el **Ir a la página modal** botón en la página principal, el

hecho clic controlador ejecuta el siguiente código:

```
esperar Navigation.PushModalAsync ( nuevo ModalPage (), cierto );
```

los **ModalPage** clase es casi idéntica a la **ModelessPage** salvo por la diferencia Título entorno y la llamada a **PopModalAsync** en el hecho clic entrenador de animales:

```
público clase ModalPage : Pagina de contenido
{
    público ModalPage ()
    {
        title = "Modal Página";

        Botón goBackButton = nuevo Botón
        {
            text = "Volver al inicio de",
            HorizontalOptions = LayoutOptions .Centrar,
            VerticalOptions = LayoutOptions .Centrar
        };
        goBackButton.Clicked += asincrono (Remitente, args) =>
        {
            esperar Navigation.PopModalAsync ();
        };

        Content = goBackButton;
    }
}
```

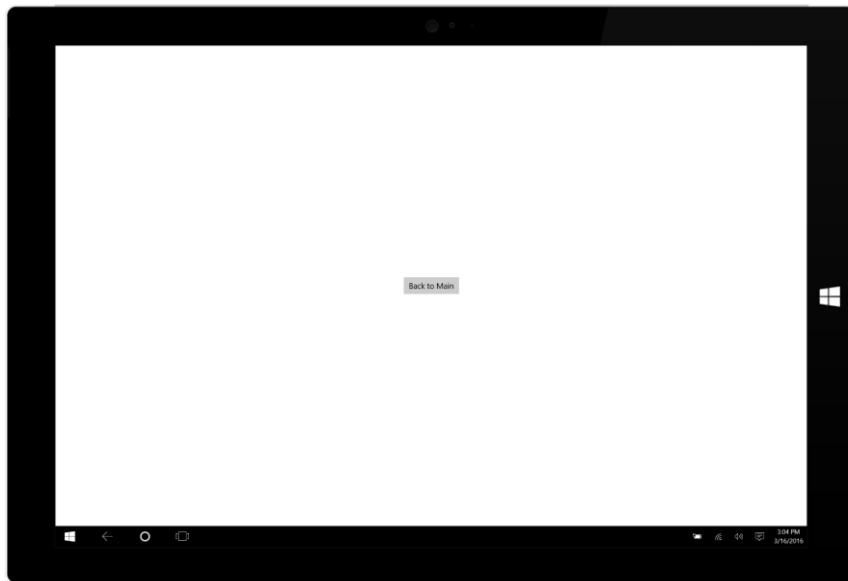
A pesar de la **Título** valor de la propiedad en la clase, ninguna de las tres plataformas de la muestra **Título** o cualquier otra interfaz de navegación de páginas:



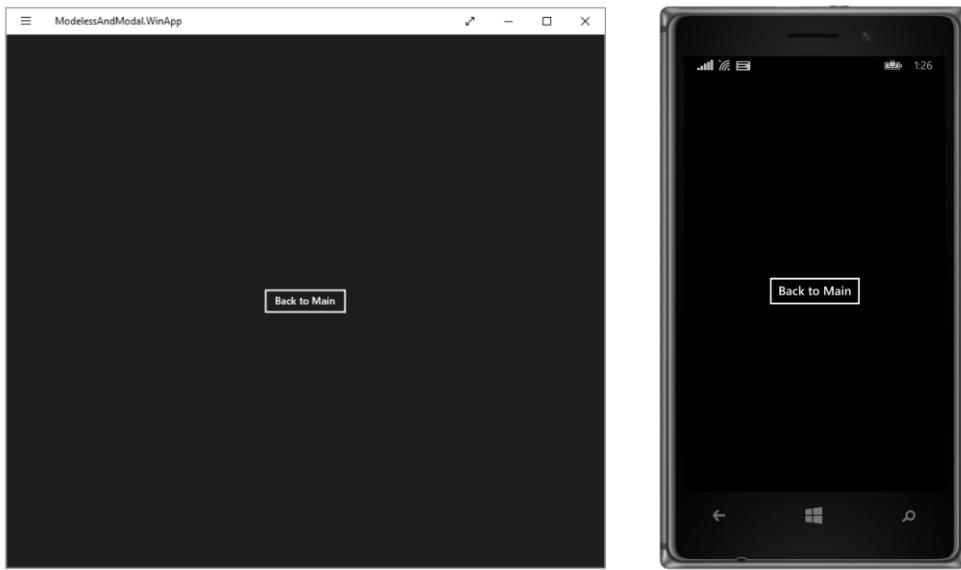
Estas pantallas ahora se parecen a las aplicaciones típicas de una sola página. Aunque no es muy obvio, tendrá una vez más que tener cuidado para evitar sobrescribir la barra de estado en la página iOS.

No es necesario el Volver al inicio de botón en las páginas Android y Windows Phone, ya que puede utilizar la Espalda botón en la parte inferior del teléfono, pero que sin duda lo necesita en la página iOS: Que **Volver al inicio de botón en el iPhone es el único camino de vuelta a Pagina principal.**

La aplicación UWP se ejecuta en Windows 10 en modo tablet no muestra un título, pero el **Espalda botón** en la esquina inferior izquierda todavía funciona para navegar de vuelta a Pagina principal.



Sin embargo, la página de Windows 8.1 necesita el **Volver al inicio de botón**, mientras que la página de Windows Phone 8.1 no lo hace porque tiene una **Espalda botón** en la parte inferior del teléfono:



Nada interna para la definición de página distingue una página no modal y una página modal. Depende de cómo la página se invoca, ya sea a través PushAsync o PushModalAsync. Sin embargo, una página en particular debe conocer la forma en que se invocó para que pueda llamar a cualquiera PopAsync o PopModalAsync para navegar de vuelta.

A lo largo del tiempo este programa se está ejecutando, sólo hay una instancia de Página principal. Continúa a seguir existiendo cuando ModelessPage y ModalPage son activos. Este es siempre el caso en una aplicación de varias páginas. Una página que llama PushAsync o PushModalAsync no dejará de existir cuando la página siguiente está activa.

Sin embargo, en este programa, cada vez que se vaya a ModelessPage o ModalPage, Se crea una nueva instancia de esa página. Cuando esa página vuelve de nuevo a Página principal, no hay más referencias a esa instancia de ModelessPage o ModalPage, y ese objeto se convierte en elegible para la recolección de basura. Esto es *no* la única manera de gestionar páginas naveables, y verá alternativas más adelante en este capítulo, pero en general lo mejor es crear una instancia de una página de la derecha antes de navegar a la misma.

Una página siempre ocupa toda la pantalla. A veces es deseable para una página modal para ocupar sólo una parte de la pantalla y de la página anterior para ser visible (pero desactivado) por debajo de esa emergente. No se puede hacer esto con páginas Xamarin.Forms. Si quieres algo por el estilo, mira la **SimpleOverlay** programa en el Capítulo 14, "disposición absoluta".

transiciones de página animadas

Los cuatro de los métodos que ha visto también están disponibles con una sobrecarga que tiene un argumento adicional de tipo bool:

Tarea PushAsync (página Página, Bool animada)

Tarea PushModalAsync (página Página, Bool animada)

Tarea <Página> PopAsync (bool animada)

Tarea <Página> PopModalAsync (bool animada)

Al establecer este argumento para cierto permite una animación de la página a la transición si una animación tal está soportado por la plataforma subyacente. Sin embargo, cuanto más simple empujar y Popular métodos permiten esta animación por defecto, por lo que sólo van a necesitar estos cuatro sobrecargas si quieras *reprimir* la animación, en cuyo caso se establece el argumento booleano para falso.

Hacia el final de este capítulo, verá un código que guarda y restaura toda la pila de navegación de páginas cuando se termina una aplicación de varias páginas. Para restaurar la pila de navegación, estas páginas se deben crear y navegar a durante el inicio del programa. En este caso, las animaciones deben ser suprimidas, y estas sobrecargas son muy útiles para ello.

También querrá para suprimir la animación si se proporciona una de sus propias animaciones de páginas de entrada, como se demuestra en el capítulo 22, "Animación".

En general, sin embargo, tendrá que utilizar las formas más simples de la empujar y Popular métodos.

variaciones visuales y funcionales

NavigationPage define varias propiedades-y varias propiedades-que enlazables adjuntas tienen el poder de cambiar la apariencia de la barra de navegación e incluso de eliminar por completo.

Puede establecer la BarBackgroundColor y BarTextColor propiedades cuando se instancia la NavigationPage en el Aplicación clase. Pruebe esto en el **ModalAndModeless** programa:

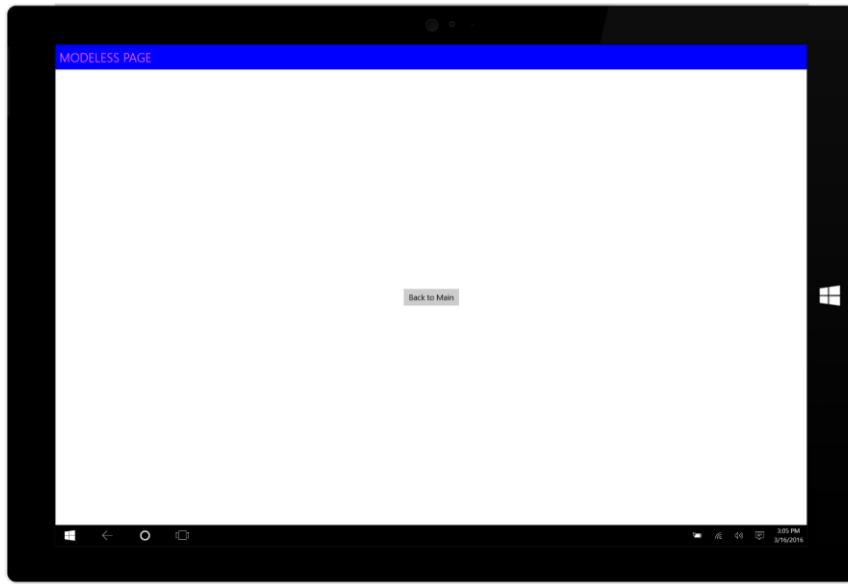
```
clase pública Aplicación : Solicitud
{
    público App ()
    {
        MainPage = nuevo NavigationPage ( nuevo Pagina principal ())
        {
            BarBackgroundColor = Color .Azul,
            BarTextColor = Color .Rosado
        };
    }
}
```

Las diversas plataformas utilizan estos colores de diferentes maneras. La barra de navegación iOS se ve afectada tanto por los colores, pero en la pantalla de Android aparece, sólo el color de fondo. Todas estas capturas de pantalla muestran

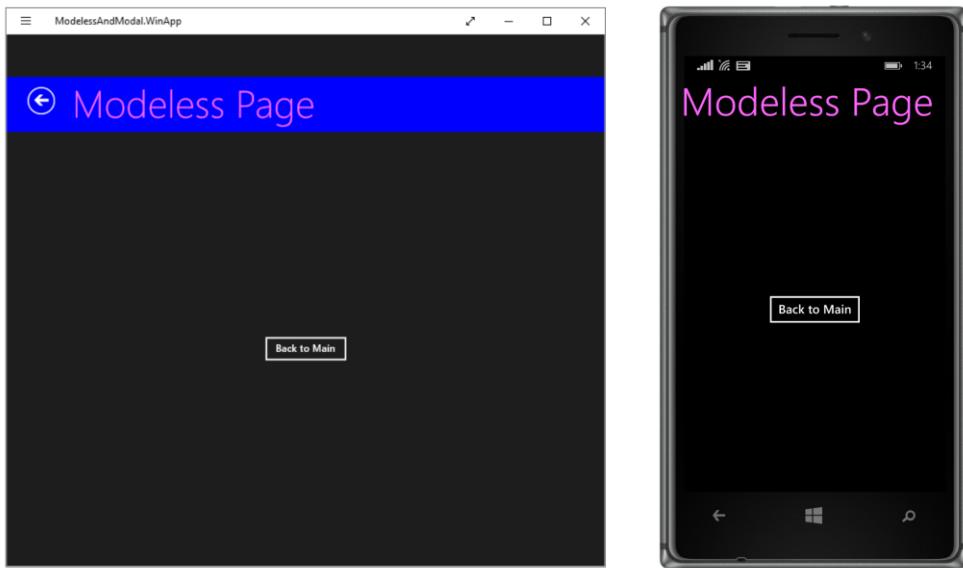
ModelessPage, pero el área de la parte superior de Pagina principal es de color de la misma manera:



La aplicación de Windows 10 en modo tablet es muy similar a la pantalla móvil de Windows 10:



Las otras dos plataformas de Windows en tiempo de ejecución también hacen uso de la BarTextColor, y las ventanas 8.1 utiliza la página BarBackgroundColor también:



los `NavigationPage` clase también define una `Tinte` propiedad, pero que la propiedad es obsoleto y debe considerarse obsoleta.

`NavigationPage` también define cuatro propiedades enlazables adjuntos que afectan a la especial Página clase en la que se establecen. Por ejemplo, supongamos que usted no desea que el `Espalda` botón que aparezca en una página no modal. He aquí cómo se establece la `NavigationPage.HasBackButton` adjunta propiedad enlazable en código en el `ModelessPage` constructor:

```
clase pública ModelessPage : Pagina de contenido
{
    público ModelessPage ()
    {
        title = "Modal Página";

        NavigationPage.SetHasBackButton (esta , falso);
        ...
    }
}
```

En XAML, que lo haría así:

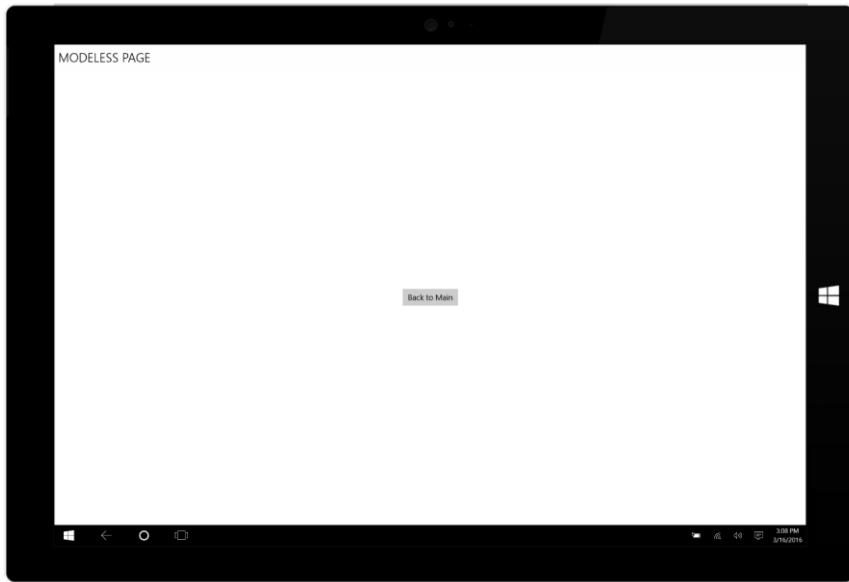
```
< Pagina de contenido xmlns = "http://xamarin.com/schemas/2014/forms"
    xmlns : X = "Http://schemas.microsoft.com/winfx/2009/xaml"
    X : Clase = "ModelessAndModal.ModelessPage"
    Título = "Modal Página"
    NavigationPage.HasBackButton = "False">
    ...
</ Pagina de contenido >
```

Y, efectivamente, cuando se desplaza a `ModelessPage`, el `Espalda` botón en la barra de navegación es

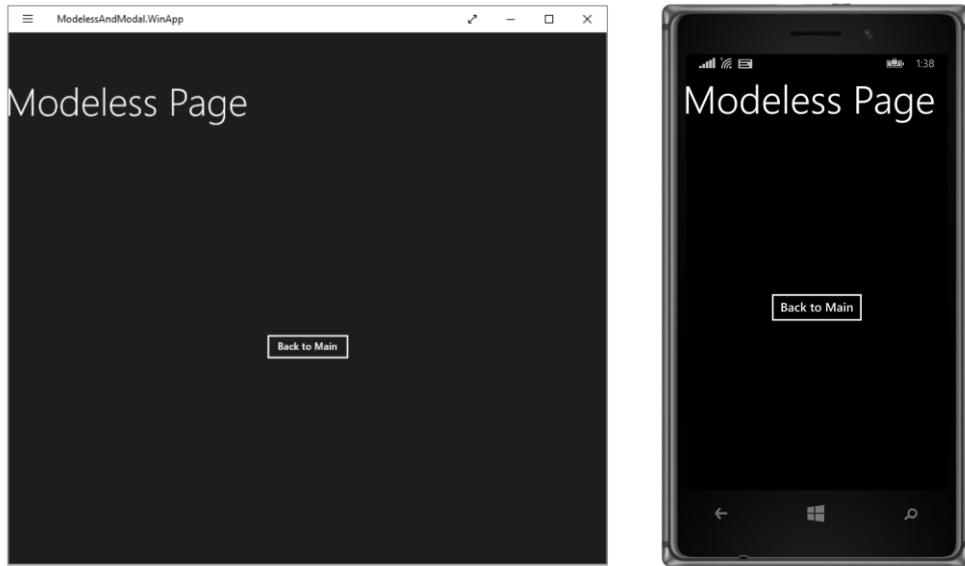
ido:



Sin embargo, un funcional **Espalda Botón** continúa existiendo en Windows 10:



los **Espalda botón** también se ha ido de la pantalla de Windows 8.1:



Una propiedad enlazable originales más extrema de `NavigationPage` elimina por completo la barra de navegación y hace que la página visualmente indistinguible de una página modal:

```
clase pública ModelessPage : Pagina de contenido
{
    público ModelessPage ()
    {
        title = "Modal Página";

        NavigationPage .SetHasNavigationBar ( esta , falso );
        ...
    }
}
```

Dos propiedades que pueden vincularse más unidos afectan el texto y el ícono en la barra de navegación. Como hemos visto, todas las plataformas muestran la `Título` propiedad en la parte superior de la página principal y una página no modal. Sin embargo, en una página no modal, la pantalla también muestra el iOS `Título` propiedad de la `anterior PAGE`- la página desde la que ha accedido a la página de modal. los `NavigationPage`

propiedad que puede vincularse adjunto puede cambiar ese texto en la página iOS. Es necesario establecer que en la página desde la que navegar a la página modal. En el `ModelessAndModal` programa, se puede establecer la propiedad de `Página principal` al igual que:

```
clase pública Pagina principal : Pagina de contenido
{
    público Pagina principal()
    {
        title = "Pagina principal";

        NavigationPage .SetBackButtonTitle ( esta , "regresa" );
        ...
    }
}
```

```
    }  
}
```

Este no afecta el título de Pagina principal en sí, sino sólo el texto que acompaña a la Espalda botón en la barra de navegación ModelessPage, y sólo en IOS. Vas a ver una captura de pantalla en breve.

La propiedad enlazable segundo adjunto es NavigationPage.TitleIcon, que sustituye al ícono de la aplicación en la barra de navegación de Android y reemplaza el título con un ícono en la página iOS. La propiedad es de tipo FileImageSource, que hace referencia a un archivo de mapa de bits en el proyecto de plataforma. En uso, que es similar a la Icono propiedad de Opción del menú y ToolbarItem.

Para que pueda experimentar con esto, algunos íconos correspondientes se han añadido a los proyectos de iOS y Android en el **ModelessAndModal** solución. Estos íconos provienen del Android **Barra de acción Icon Pack** discutido en el capítulo 13, "Los mapas de bits." (En el capítulo 13, busque la sección "mapas de bits específicos de la plataforma," y luego "Barras de herramientas y sus íconos," y finalmente "Los íconos de Android.")

Para iOS, los íconos son de la **ActionBarIcons / holo_light / 08_camera_flash_on** directorio. Estos íconos muestran un rayo. Las imágenes de la **MDPI**, **xdpi**, y **xxdpi** directorios son 32, 64, y 96 píxeles cuadrados, respectivamente. Dentro de recursos carpeta del proyecto IOS, el mapa de bits cuadrado de 32 píxeles tiene el nombre original de **ic_action_flash_on.png**, y los dos más grandes archivos fueron renombrados con **@2x** y **3x** sufijos, respectivamente.

Para Android, los íconos son de la **ActionBarIcons / holo_dark / 08_camera_flash_on** directorio; estos son los primeros planos blancos sobre fondos transparentes. Los archivos de la **MDPI**, **IPAP**, **xdpi**, y **xxdpi** directorios se añadieron al proyecto Android.

Puede mostrar estos íconos en la página modal añadiendo el código siguiente a la Modeless-Página constructor:

```
    público clase ModelessPage : Pagina de contenido  
{  
    público ModelessPage ()  
    {  
        title = "Modal Página";  
  
        Si ( Dispositivo.os == TargetPlatform.iOS || Dispositivo.os == TargetPlatform.Android)  
            NavigationPage .SetTitleIcon ( esta , "ic_action_flash_on.png" );  
        ...  
    }  
}
```

Aquí está ModelessPage tanto con el alternativo Espalda el texto del botón de "volver" situado en Pagina principal y los íconos conjunto de ModelessPage:



Como se puede ver, el ícono sustituye a la normalidad Título texto en la página iOS.

Ni el `NavigationPage.BackButtonTitle` ni el `NavigationPage.TitleIcon` propiedad que puede vincularse adjunta afecta a ninguna de las plataformas de Windows o Windows Phone.

Los programadores familiarizados con la arquitectura de Android son a veces curioso como la navegación de páginas Xamarin.Forms se integra con el aspecto de la arquitectura de aplicaciones de Android conocida como la *actividad*. Una aplicación Xamarin.Forms se ejecuta en un dispositivo Android comprende una sola actividad, y la navegación por la página está construida encima de eso. UN Página de contenido Es un objeto Xamarin.Forms; no es una actividad Android, o un fragmento de una actividad.

Explorar la mecánica

Como hemos visto, la empujar y Popular métodos devuelven Tarea objetos. En general vamos a usar esperar cuando se llama a estos métodos. Aquí está la llamada a `PushAsync` en el Página principal clase de `ModelessAndModal`:

```
esperar Navigation.PushAsync ( nuevo ModelessPage );
```

Supongamos que tiene un código después de esta declaración. Cuando se alcanza a ejecutar ese código? Sabemos que se ejecuta cuando el `PushAsync` tarea se completa, pero cuando es eso? Es que después de que el usuario ha aprovechado una

Espalda botón de `ModelessPage` para volver de nuevo a ¿Página principal?

No, ese no es el caso. Los `PushAsync` tarea se completa con bastante rapidez. La finalización de esta tarea no indica que el proceso de navegación de páginas se ha completado, pero hace indicar cuándo es seguro para obtener el estado actual de la pila de páginas de navegación.

Siguiendo un `PushAsync` o `PushModalAsync` llama, se producen los siguientes eventos. Sin embargo, el orden preciso en que se producen estos eventos es dependiente de la plataforma:

- La vocación página PushAsync o PushModalAsync generalmente recibe una llamada a su OnDisappearing anular.
- La página está navegando a crear una llamada a su OnAppearing anular.
- Los PushAsync o PushModalAsync tarea se completa.

Para repetir: El orden en que ocurren estos eventos depende de la plataforma y también de si la navegación es a una página o una página no modal modal.

Siguiendo un PopAsync o PopModalAsync llaman, se producen los siguientes eventos, de nuevo en un orden que depende de la plataforma:

- La vocación página PopAsync o PopModalAsync recibe una llamada a su OnDisappearing anular.
- La página que se devuelve a lo general recibe una llamada a su OnAppearing anular.
- Los PopAsync o PopModalAsync rendimientos de trabajo.

Se dará cuenta de dos usos de la palabra "generalmente" en esas descripciones. Esta palabra se refiere a una excepción a estas reglas cuando un dispositivo Android navega a una página modal. La página que llama PushModalA-sincronizar no recibe una llamada a su OnDisappearing anular, y esa misma página no recibe una llamada a su OnAppearing Intrusión al de las llamadas páginas modal PopModalAsync.

Además, las llamadas al OnDisappearing y OnAppearing anulaciones no indican necesariamente la navegación de páginas. En iOS, la OnDisappearing anulación se llama en la página activa cuando el programa termina. En la plataforma Windows Phone Silverlight (que ya no es apoyado por Xamarin.Forms), una página recibida OnDisappearing llama cuando el usuario invoca una Picker, DatePicker, o

TimePicker en la pagina. Por estas razones, el OnDisappearing y OnAppearing anulaciones no puede ser tratada como indicaciones de navegación de páginas garantizados, aunque hay momentos en los que se deben utilizar para tal fin.

los INavigation interfaz que define estos empujar y Popular pide, asimismo, define dos propiedades que proporcionan acceso a la pila de navegación real:

- NavigationStack, que contiene las páginas modal
- ModalStack, que contiene las páginas modales

los conjunto descriptores de acceso de estas dos propiedades no son públicas, y las propiedades mismas son de tipo IReadOnlyList <Página>, por lo que no se puede modificar directamente. (Como se verá, los métodos están disponibles para modificar la pila de página de una manera más estructurada.) A pesar de que estas propiedades no se implementan con Pila <T> clases, funcionan como una pila de todos modos. El elemento de la IReadOnlyList con un índice de cero es la página más antigua, y el último elemento es la página más reciente.

La existencia de estas dos colecciones de páginas no modales y modales sugiere que no modal y la navegación por la página modal no pueden mezclarse, y esto es cierto: Una página modal puede navegar a una página modal, pero una página modal *no poder* navegar a una página no modal.

Alguna experimentación revela que la Navegación propiedad de diferentes instancias página mantiene diferentes colecciones de la pila de navegación. (En particular, después de la navegación a una página modal, la la Navegación navigationStack asociado a esa página modal está vacía.) El enfoque más infalible es trabajar con las instancias de estas colecciones mantenidas por el Navegación propiedad de la NavigationPage ejemplo establecido en la Pagina principal propiedad de la Aplicación clase.

Con cada llamada a PushAsync o PopAsync, el contenido de la NavigationStack cambiar, ya sea una nueva página se añade a la colección o una página se quita de la colección. Del mismo modo, con cada llamada a PushModalAsync o PopModalAsync, el contenido de la ModalStack cambio.

La experimentación revela que no es seguro de usar el contenido de la NavigationStack o ModalStack durante las llamadas a la OnAppearing o OnDisappearing anula mientras que la navegación de la página está en curso. El único método que funciona para todas las plataformas es esperar hasta que el PushAsync, PushModalAsync, PopAsync, o PopModalAsync tarea se completa. Esa es su indicación de que estas colecciones de pila son estables y precisos.

los NavigationPage clase también define una propiedad única para hacerse con nombre Página actual. Esta instancia de página es el mismo que el último elemento de la NavigationStack colecciones disponibles a partir Navegación-Página. Sin embargo, cuando una página modal está activo, Página actual continúa para indicar el último *modal*/ la página que estaba activa antes de la navegación a una página modal.

Vamos a explorar los detalles y la mecánica de navegación de páginas con un programa llamado **SinglePageNavigation**, así llamada porque el programa contiene clase sólo una página, llamada SinglePageNavigationPage. El programa se desplaza entre varias instancias de este una clase.

Uno de los propósitos de la **SinglePageNavigation** programa es para prepararse para escribir una aplicación que guarda la pila de navegación cuando la aplicación está suspendida o terminada, y para restaurar la pila cuando se reinicia la aplicación. Hacer esto depende de la capacidad de la aplicación para extraer información confiable de la NavigationStack y ModalStack propiedades.

Aquí está el archivo XAML para el SinglePageNavigationPage clase:

```
< Pagina de contenido xmlns = "http://xamarin.com/schemas/2014/forms"
    xmlns : X = "http://schemas.microsoft.com/winfx/2009/xaml"
    X : Clase = "SinglePageNavigation.SinglePageNavigationPage"
    X : Nombre = "Página">

    < StackLayout >
        < StackLayout.Resources >
            < ResourceDictionary >
                < Estilo X : Llave = "BaseStyle" Tipo de objetivo = "Ver">
                    < Setter Propiedad = "VerticalOptions" Valor = "CenterAndExpand" />
                </ Estilo >

                < Estilo Tipo de objetivo = "Botón" Residencia en = "( StaticResource baseStyle ) ">
                    < Setter Propiedad = "HorizontalOptions" Valor = "Center" />
                </ Estilo >

                < Estilo Tipo de objetivo = "Etiqueta" Residencia en = "( StaticResource baseStyle ) ">
```

```

        < Setter Propiedad = "HorizontalTextAlignment" Valor = "Center" />
    </ Estilo >
</ ResourceDictionary >
</ StackLayout.Resources >

< Etiqueta Texto = "{ Unión Fuente = { X : Referencia página }, Camino Título = }" >

< Botón X : Nombre = "ModelessGoToButton"
    Texto = "Ir a la Página no modal"
    hecho clic = "OnGoToModelessClicked" />

< Botón X : Nombre = "ModelessBackButton"
    Texto = "Vuelta de página no modal"
    hecho clic = "OnGoBackModelessClicked" />

< Botón X : Nombre = "ModalGoToButton"
    Texto = "Ir a la Página modal"
    hecho clic = "OnGoToModalClicked" />

< Botón X : Nombre = "ModalBackButton"
    Texto = "Vuelta de página modal"
    hecho clic = "OnGoBackModalClicked" />

< Etiqueta X : Nombre = "CurrentPageLabel"
    Texto ="" />

< Etiqueta X : Nombre = "ModelessStackLabel"
    Texto ="" />

< Etiqueta X : Nombre = "ModalStackLabel"
    Texto ="" />
</ StackLayout >
</ Página de contenido >

```

El archivo XAML crea una instancia de cuatro Botón y cuatro Etiqueta elementos. El primero Etiqueta tiene un enlace de datos para mostrar la página de Título propiedad para que el título es visible, independientemente de la plataforma y si la página es modal o no. Los cuatro botones son para navegar hacia y desde las páginas modal o modal. Las tres etiquetas restantes muestran otro conjunto de información de código.

Aquí es más o menos la primera mitad del archivo de código subyacente. Observe el código de constructor que establece el Título propiedad para el texto "Página #", donde el signo almohadilla indica un número que comienza en cero para la primera página instanciada. Cada vez que se crea una instancia de esta clase, ese número se incrementa:

```

pública clase parcial SinglePageNavigationPage : Página de contenido
{
    static int count = 0;
    static bool firstPageAppeared = falso ;
    estático cadena de sólo lectura separador = nueva cadena ('.', 20);

    pública SinglePageNavigationPage ()
    {
        InitializeComponent ();
    }
}

```

```
// Establecer título de ejemplo de base cero de esta clase.
title = "Página " + Count++;
}

vacío asíncrono OnGoToModelessClicked ( objeto remitente, EventArgs args)
{
    SinglePageNavigationPage newPassword = nuevo SinglePageNavigationPage ();
    Depurar .WriteLine (separador);
    Depurar .Línea de escritura( "Llamando PushAsync de {0} a {1}" , esta , nueva pagina);
    esperar Navigation.PushAsync (newPage);
    Depurar .Línea de escritura( "PushAsync completó" );

    // Muestra la información de la página pila en esta página.
    newPassword.DisplayInfo ();
}

vacío asíncrono OnGoToModalClicked ( objeto remitente, EventArgs args)
{
    SinglePageNavigationPage newPassword = nuevo SinglePageNavigationPage ();
    Depurar .WriteLine (separador);
    Depurar .Línea de escritura( "Llamando PushModalAsync de {0} a {1}" , esta , nueva pagina);
    esperar Navigation.PushModalAsync (newPage);
    Depurar .Línea de escritura( "PushModalAsync completó" );

    // Muestra la información de la página pila en esta página.
    newPassword.DisplayInfo ();
}

vacío asíncrono OnGoBackModelessClicked ( objeto remitente, EventArgs args)
{
    Depurar .WriteLine (separador);
    Depurar .Línea de escritura( "Llamando PopAsync de {0}" , esta );
    Página page = esperar Navigation.PopAsync ();
    Depurar .Línea de escritura( "PopAsync completado y devuelto {0}" , página);

    // Muestra la información de la página pila en la página de ser devuelto a.
    NavigationPage navPage = ( NavigationPage ) Aplicación .Current.MainPage;
    (( SinglePageNavigationPage ) NavPage.CurrentPage) .DisplayInfo ();
}

vacío asíncrono OnGoBackModalClicked ( objeto remitente, EventArgs args)
{
    Depurar .WriteLine (separador);
    Depurar .Línea de escritura( "Llamando PopModalAsync de {0}" , esta );
    Página page = esperar Navigation.PopModalAsync ();
    Depurar .Línea de escritura( "PopModalAsync completado y devuelto {0}" , página);

    // Muestra la información de la página pila en la página de ser devuelto a.
    NavigationPage navPage = ( NavigationPage ) Aplicación .Current.MainPage;
    (( SinglePageNavigationPage ) NavPage.CurrentPage) .DisplayInfo ();
}

protegido override void OnAppearing ()
{
```

```

base.OnAppearing ();
Depurar.Línea de escritura("0 OnAppearing", Título);

Si (! FirstPageAppeared)
{
    DisplayInfo ();
    firstPageAppeared = cierto ;
}
}

protegido override void OnDisappearing ()
{
    base.OnDisappearing ();
    Depurar.Línea de escritura("0 OnDisappearing", Título);
}

// Identificar cada caso por su título.

público cadena de anulación Encadenar()
{
    regreso Título;
}
...
}

```

Cada una de las hecho clic manejadores de los cuatro botones de muestra información utilizando Debug.WriteLine. Cuando se ejecuta el programa en el depurador de Visual Studio o Xamarin de estudio, este texto aparece en el Salida ventana.

El archivo de código subyacente también anula la OnAppearing y OnDisappearing métodos. Estos son importantes, por lo general le dicen cuando la página se está navegando a (OnAppearing) o navegado a partir de (OnDisappearing).

Sin embargo, como se mencionó anteriormente, Android es un poco diferente: Una página de Android que llama PushModalAsync no recibe una llamada a su OnDisappearing método, y cuando la página modal regresa a esa página, la página original no recibe una llamada correspondiente a su OnAppearing método. Es como si la página se queda en segundo plano mientras se visualiza la página modal, y eso es mucho el caso: Si se vuelve a ModelessAndModal y establecer el Color de fondo La página modal Color.From-
RGBA (0, 0, 0, 0,5), se puede ver la página anterior detrás de la página modal. Pero esto es sólo el caso de Android.

Todos hecho clic manipuladores de SinglePageNavigationPage hacer una llamada a un método llamado displayInfo. Este método se muestra a continuación y muestra información sobre el NavigationStack y ModalStack -incluyendo las páginas en las pilas y la Página actual propiedad mantenida por el NavigationPage objeto.

Sin embargo, estos hecho clic manipuladores hacen *no* llama a DisplayInfo método en la instancia actual de la página debido a que la hecho clic manipuladores están efectuando una transición a otra página. Los hecho clic manipuladores deben llamar al DisplayInfo método en la instancia de página a la que están navegando.

Las llamadas a DisplayInfo en el hecho clic manipuladores que llaman PushAsync y PushModalAsync son

fácil porque cada uno hecho clic manejador ya tiene la nueva instancia de página que se está navegado. Las llamadas a DisplayInfo en el hecho clic manipuladores que llaman PopAsync y PopModalAsync son un poco más difícil, ya que necesitan para obtener la página de ser devuelto a. Este no es el Página instancia volvió de la PopAsync y PopModalAsync Tareas. Ese Página instancia resulta ser la misma página que llama a estos métodos.

En cambio, el hecho clic manipuladores que llaman PopAsync y PopModalAsync obtener la página que se volvió a Del Página actual propiedad de NavigationPage:

```
NavigationPage navPage = ( NavigationPage ) Aplicación .Current.MainPage;
(( SinglePageNavigationPage ) NavPage.CurrentPage) .DisplayInfo();
```

Lo que es crucial es que el código para obtener este nuevo Página actual propiedad y las llamadas a displayInfo todo ocurre después el asíncrona empujar o Popular tarea se ha completado. Fue entonces cuando esta información se convierte en válido.

sin embargo, el DisplayInfo método también se debe llamar al iniciar el programa por primera vez. Como se verá, DisplayInfo hace uso de la Pagina principal propiedad de la Aplicación clase para obtener el Navegación-Página instanciado en el Aplicación constructor. Sin embargo, esa Pagina principal la propiedad aún no se ha establecido en el Aplicación cuando el constructor SinglePageNavigationPage se ejecuta el constructor, por lo que el constructor de página no puede llamar DisplayInfo. En cambio, el OnAppearing anulación hace esa llamada, pero sólo para la primera instancia de página:

```
Si ( ! FirstPageAppeared )
{
    DisplayInfo ();
    firstPageAppeared = cierto ;
}
```

Además de mostrar el valor de Página actual y el NavigationStack y ModalStack colecciones, la DisplayInfo método también activa y desactiva los cuatro Botón elementos de la página de modo que siempre es legal para presionar una habilitado Botón.

A continuación se DisplayInfo y los dos métodos que utiliza para mostrar las colecciones de pila:

```
público clase parcial SinglePageNavigationPage : Pagina de contenido
{
    ...
    public void DisplayInfo ()
    {
        // Obtener el NavigationPage y mostrar su propiedad currentPage.
        NavigationPage navPage = ( NavigationPage ) Aplicación .Current.MainPage;

        currentPageLabel.Text = Cuerda .Formato( "NavigationPage.CurrentPage = {0}" ,
                                              navPage.CurrentPage);

        // Obtener las pilas de navegación de la NavigationPage.
        IReadOnlyList < Página > NavStack = navPage.Navigation.NavigationStack;
        IReadOnlyList < Página > ModStack = navPage.Navigation.ModalStack;

        // muestra los recuentos y contenido de estas pilas.
```

```

En t modelessCount = navStack.Count;
En t modalCount = modStack.Count;

modelessStackLabel.Text = Cuerda .Formato( "NavigationStack tiene {0} página de {1} {2}" ,
                                         modelessCount,
                                         modelessCount == 1? "" : "S",
                                         ShowStack (navStack));

modalStackLabel.Text = Cuerda .Formato( "ModalStack tiene {0} página de {1} {2}" ,
                                         modalCount,
                                         modalCount == 1? "" : "S",
                                         ShowStack (modStack));

// Activar y desactivar los botones basados en los conteos.

bool noModals = modalCount == 0 || (ModalCount == 1 && modStack [0] es NavigationPage );

modelessGoToButton.IsEnabled = noModals;
modelessBackButton.IsEnabled = modelessCount > 1 && noModals;
! = modalBackButton.IsEnabled noModals;
}

cuerda ShowStack ( IReadOnlyList < Página > Pila de páginas)
{
    Si (PageStack.Count == 0)
        regreso "";

    StringBuilder constructor = nuevo StringBuilder ();

    para cada ( Página página en pila de páginas)
    {
        builder.Append (builder.Length == 0? "(" : " ");
        builder.Append (StripNamespace (página));
    }

    builder.Append ( ")");
    regreso builder.ToString ();
}

cuerda StripNamespace ( Página página)
{
    cuerda páginestring = page.ToString ();

    Si (pageString.Contains ( ))
        páginestring = pageString.Substring (pageString.LastIndexOf ( ) + 1);

    regreso páginestring;
}
}

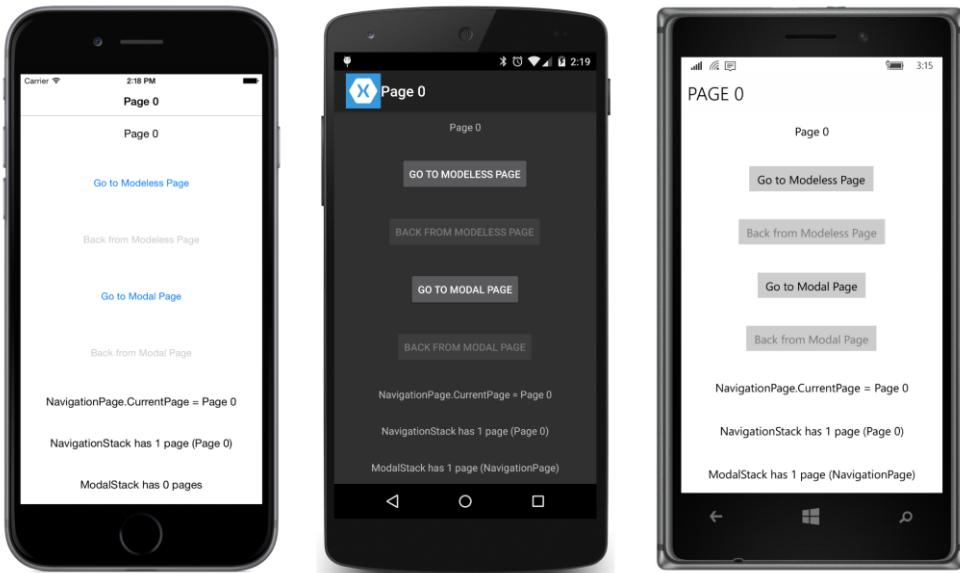
```

Parte de la lógica que implica la habilitación e inhabilitación de la Botón elementos se pondrán de manifiesto cuando vea algunas de las pantallas que el programa muestra. Siempre se puede comentar que la activación y desactivación de código para explorar lo que sucede cuando se presiona un inválido Botón.

Las reglas generales son las siguientes:

- Una página modal puede navegar a otra página no modal o una página modal.
- Una página modal sólo se puede navegar a otra página modal.

La primera vez que ejecute el programa, verá el siguiente. El XAML incluye una pantalla de la Título propiedad en la parte superior, por lo que es visible en todas las páginas:



El tres Etiqueta elementos en la parte inferior de la página de visualización del Página actual propiedad de la NavigationPage objeto y la NavigationStack y ModalStack, tanto obtenida a partir de la NAV igation propiedad de la NavigationPage.

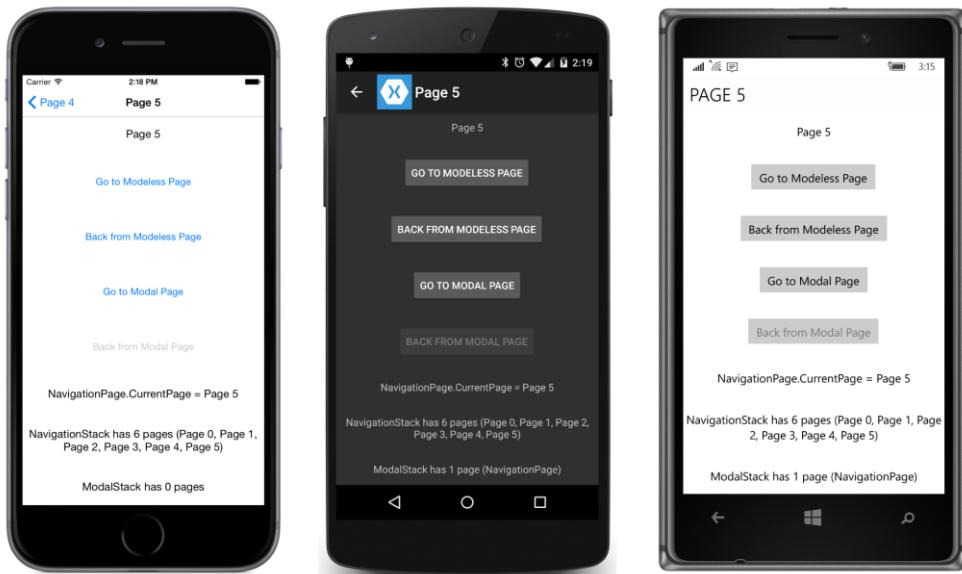
En las tres plataformas, la NavigationStack contiene un artículo, que es la página principal. El contenido de la ModalStack, sin embargo, varía de plataforma. En las plataformas Android y Windows en tiempo de ejecución, la pila modal contiene un artículo (el NavigationPage objeto), pero la pila está vacía modal en IOS.

Esta es la razón por la DisplayInfo método establece el noModels variable booleana a cierto Si bien la pila modal tiene un recuento de cero o si contiene un solo elemento, pero que el tema es NavigationPage:

```
bool noModels = modalCount == 0 || (ModalCount == 1 && modStack [0] es NavigationPage );
```

Observe que el Página actual la propiedad y el artículo en NavigationStack no son instancias de NavigationPage, pero en cambio son instancias de SinglePageNavigationPage, que se deriva de Pagina de contenido. Es SinglePageNavigationPage que define su Encadenar método para visualizar el título de la página.

Ahora pulse el Ir a la Página no modal botón cinco veces y esto es lo que verá. Las pantallas son consistentes a un lado de la pila modal en la pantalla de iOS:



Tan pronto como se pulsa el **Ir a la Página no modal** botón una vez, la **De vuelta de página no modal** botón está habilitado. La lógica es la siguiente:

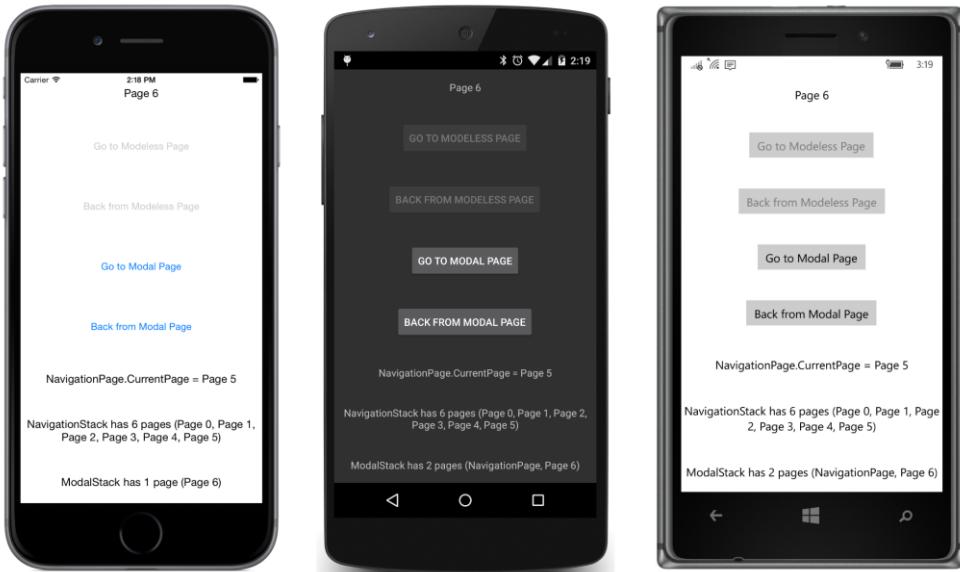
```
modelessBackButton.IsEnabled = modelessCount > 1 && noModals;
```

En la llanura Inglés, la **De vuelta de página no modal** botón debe estar habilitado si hay al menos dos elementos de la pila no modal (la página original y la página actual) y si la página actual no es una página modal.

Si pulsa que **A la vuelta de ModelessPage** botón en este punto, verá el NavigationStack disminuye de tamaño hasta llegar de nuevo a **Página 0**. Durante el **Página actual** propiedad sigue indicando el último elemento NavigationStack.

Si a continuación pulsa **Ir a la Página no modal** de nuevo, se verá más elementos añadidos a la Navegación-Apilar con cada vez mayores números de página ya que los nuevos SinglePageNavigationPage objetos se instancian.

En su lugar, intente presionar el **Ir a la página modal** botón:



Ahora ModalStack contiene la nueva página, pero Página actual Todavía se refiere a la última página no modal. La pila modal iOS sigue desaparecido esa inicial NavigationPage objeto presente en las otras plataformas.

Si a continuación pulsa De vuelta de página modal, la pila modal se restaura correctamente a su estado inicial.

aplicaciones de varias páginas por lo general tratan de salvar la pila de navegación cuando están suspendidos o terminados, y luego restaurar esa pila cuando empiezan de nuevo. Hacia el final de este capítulo, verá código que utiliza NavigationStack y ModalStack para hacer ese trabajo.

modalidad de hacer cumplir

En general, sus aplicaciones se utilizará probablemente no modales páginas, excepto en circunstancias especiales cuando la aplicación necesita obtener información crucial del usuario. La aplicación puede mostrar una página modal que el usuario no puede descartar hasta que se haya introducido esta información crucial.

Un pequeño problema, sin embargo, es que un usuario de un teléfono Android o Windows siempre puede volver a la página anterior pulsando el estándar **Espalda botón** en el dispositivo. Para hacer cumplir modalidad, para asegurarse de que el usuario introduce la información deseada antes de salir de la página de la aplicación, debe desactivar ese botón.

Esta técnica se demostró en el **Modal Enforcement** programa. La página principal consiste únicamente en una Botón:

```
< Página de contenido xmlns = "http://xamarin.com/schemas/2014/forms"
      xmlns : X = "Http://schemas.microsoft.com/winfx/2009/xaml"
      X : Clase = "ModalEnforcement.ModalEnforcementHomePage"
      Título = "Página Principal">
```

```
< Botón Texto = "Ir a la Página modal"
    HorizontalOptions = "Centro"
    VerticalOptions = "Centro"
    hecho clic = "OnGoToButtonClicked" />

</ Pagina de contenido >
```

maneja el archivo de código subyacente del hecho clic evento del botón mediante la navegación a una página modal:

```
público clase parcial ModalEnforcementHomePage : Pagina de contenido
{
    público ModalEnforcementHomePage ()
    {
        InitializeComponent ();
    }

    vacío asíncrono OnGoToButtonClicked ( objeto remitente, EventArgs args )
    {
        esperar Navigation.PushModalAsync ( nuevo ModalEnforcementModalPage () );
    }
}
```

El archivo XAML para el ModalEnforcementModalPage contiene dos Entrada elementos, una Recogedor elemento, y una Botón etiquetado Hecho. El margen de beneficio es más extenso de lo que podría anticipar ya que contiene una MultiTrigger para establecer el Está habilitado propiedad del botón de Certo sólo si algo se ha escrito en los dos Entrada elementos y algo también se ha introducido en el Recogedor. Esta

MultiTrigger Requiere tres ocultos Cambiar elementos, utilizando una técnica discuten en el Capítulo 23, "disparadores y comportamientos":

```
< Pagina de contenido xmlns = "http://xamarin.com/schemas/2014/forms"
    xmlns : X = "Http://schemas.microsoft.com/winfx/2009/xaml"
    X : Clase = "ModalEnforcement.ModalEnforcementModalPage"
    Título = "Modal Página">

    < StackLayout Relleno = "20, 0">
        < Entrada X : Nombre = "Entry1"
            Texto = ""
            marcador de posición = "Entrar nombre"
            VerticalOptions = "CenterAndExpand" />

        <! - Interruptor invisible para ayudar con la lógica MultiTrigger ->
        < Cambiar X : Nombre = "Switch1" Es visible = "False" >
            < Switch.Triggers >
                < DataTrigger Tipo de objetivo = "Switch"
                    Unión = "{ Unión Fuente = { X : Referencia entry1 }, Camino = Text.Length}"
                    Valor = "0" >
                    < Setter Propiedad = "IsToggled" Valor = "True" />
                </ DataTrigger >
            </ Switch.Triggers >
        </ Cambiar >

        < Entrada X : Nombre = "Entrada2"
            Texto = ""
```

```
marcador de posición = "Introducir dirección de correo electrónico"
VerticalOptions = "CenterAndExpand" />

<! - Interruptor invisible para ayudar con la lógica MultiTrigger ->
< Cambiar X : Nombre = "Switch2" Es visible = "False" >
< Switch.Triggers >
< DataTrigger Tipo de objetivo = "Switch" 
    Unión = "{ Unión Fuente = { X : Referencia entrada2 }, Camino = Text.length}"
    Valor = "0">
    < Setter Propiedad = "IsToggled" Valor = "True" />
</ DataTrigger >
</ Switch.Triggers >
</ Cambiar >

< Recogedor X : Nombre = "Selector" 
    Título = "Lenguaje de programación favorito"
    VerticalOptions = "CenterAndExpand" >
< Picker.Items >
< X : Cuerda > DO# </X : Cuerda >
< X : Cuerda > F# </X : Cuerda >
< X : Cuerda > C objetivo </X : Cuerda >
< X : Cuerda > Rápido </X : Cuerda >
< X : Cuerda > Java </X : Cuerda >
</ Picker.Items >
</ Recogedor >

<! - Interruptor invisible para ayudar con la lógica MultiTrigger ->
< Cambiar X : Nombre = "Switch3" Es visible = "False" >
< Switch.Triggers >
< DataTrigger Tipo de objetivo = "Switch" 
    Unión = "{ Unión Fuente = { X : Referencia recogedor }, Camino = SelectedIndex}"
    Valor = ". 1">
    < Setter Propiedad = "IsToggled" Valor = "True" />
</ DataTrigger >
</ Switch.Triggers >
</ Cambiar >

< Botón X : Nombre = "DoneButton" 
    Texto = "Done"
    Está habilitado = "False"
    HorizontalOptions = "Centro"
    VerticalOptions = "CenterAndExpand"
    hecho clic = "OnDoneButtonClicked">
< Button.Triggers >
< MultiTrigger Tipo de objetivo = "Button" >
< MultiTrigger.Conditions >
< BindingCondition Unión = "{ Unión Fuente = { X : Referencia switch1 },
    Camino = IsToggled}" 
    Valor = "False" />
< BindingCondition Unión = "{ Unión Fuente = { X : Referencia switch2 },
    Camino = IsToggled}" 
    Valor = "False" />
```

```

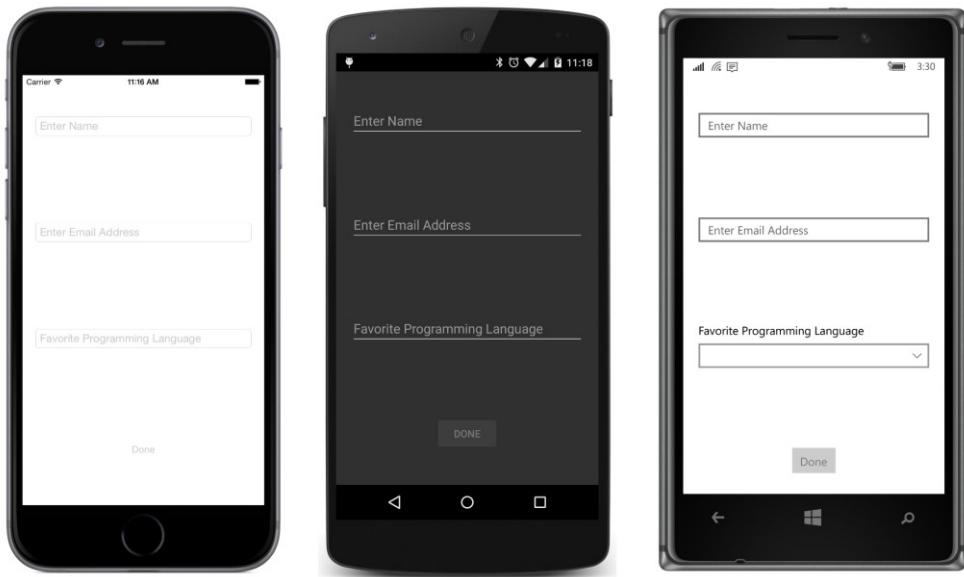
< BindingCondition Unión = "{ Unión Fuente = { X : Referencia switch3 },
    Camino = IsToggled}">
    Valor = "False" />
</ MultiTrigger.Conditions >

< Setter Propiedad = "IsEnabled" Valor = "True" />
</ MultiTrigger >
</ Button.Triggers >
</ Botón >
</ StackLayout >
</ Página de contenido >

```

En un programa de la vida real, no habría probablemente también ser un cheque que la dirección de correo electrónico es válida. La lógica simple en el archivo XAML simplemente comprueba la presencia de al menos un carácter.

Esta es la página de modal tal como aparece en primer lugar, cuando nada se ha introducido. Observe que el **Hecho** botón está desactivado:



Normalmente, el usuario puede pulsar la norma **Espalda** botón en la parte inferior izquierda de las pantallas de los teléfonos Android y Windows para volver de nuevo a la página principal. Para inhibir el comportamiento normal de la **Espalda** botón, la página modal debe anular lo virtual OnBackPressed método. Puede proporcionar su propia **Espalda** procesamiento en este botón de anulación y devolución cierto. Para desactivar por completo la **Espalda** botón, devuelva cierto sin hacer nada más. Para permitir que el defecto **Espalda** procesamiento de botón que se produzca, llama a la implementación de la clase base. He aquí cómo el archivo de código subyacente de ModalEnforcementMod-

ALPAGE lo hace:

```

público clase parcial ModalEnforcementModalPage : Página de contenido
{
    público ModalEnforcementModalPage ()

```

```

{
    InitializeComponent ();
}

protegido bool anulación OnBackButtonPressed ()
{
    Si (DoneButton.IsEnabled)
    {
        base de retorno .OnBackButtonPressed ();
    }
    return true ;
}

vacio asincrónico OnDoneButtonClicked ( objeto remitente, EventArgs args)
{
    esperar Navigation.PopModalAsync ();
}
}
}

```

Sólo si el **Hecho** botón en el archivo XAML está habilitada la OnBackButtonPressed anular llamar a la implementación de la clase base del método y devolver el valor que se devuelve desde que la implementación. Esto hace que la página modal para volver a la página que lo invocó. Si el **Hecho** botón está, entonces los rendimientos de anulación con discapacidad cierto lo que indica que se ha terminado de realizar todas las manipulaciones que se desea para el **Espalda** botón.

los hecho clic controlador para el Hecho botón llama simplemente PopModalAsync, como siempre.

los Página clase también define una **SendBackPressed** que hace que el **OnBackButtonPressed** método que se llama. Debería ser posible implementar el hecho clic controlador para el **Hecho** botón llamando a este método:

```

vacío OnDoneButtonClicked ( objeto remitente, EventArgs args)
{
    SendBackPressed ();
}

```

Aunque esto funciona en iOS y Android, que actualmente no funciona en las plataformas de Windows en tiempo de ejecución.

En la programación del mundo real, es más probable que se va a utilizar un modelo de vista para acumular la información que el usuario entra en la página de modal. En ese caso, el propio modelo de vista puede contener una propiedad que indica si toda la información introducida es válida.

los MvvmEnforcement programa utiliza esta técnica, e incluye un pequeño modelo de vista-nombre apropiado **LittleViewModel**:

```

espacio de nombres MvvmEnforcement
{
    clase pública LittleViewModel : INotifyPropertyChanged
    {
        cuerda nombre Correo Electronico;
        cuerda [] = {Idiomas "DOI#", "F#", "C objetivo", "Rápido", "Java"};
    }
}

```

```
En t languageIndex = -1;
bool es válida;

evento público PropertyChangedEventHandler PropertyChanged;

public string Nombre
{
    conjunto
    {
        Si (Nombre! = valor )
        {
            name = valor ;
            OnPropertyChanged ( "Nombre" );
            TestIfValid ();
        }
    }
    obtener { regreso nombre; }
}

public string Email
{
    conjunto
    {
        Si (Correo electrónico! = valor )
        {
            email = valor ;
            OnPropertyChanged ( "Email" );
            TestIfValid ();
        }
    }
    obtener { regreso correo electrónico; }
}

público IEnumerable < cuerda > Idiomas
{
    obtener { regreso idiomas; }
}

public int LanguageIndex
{
    conjunto
    {
        Si (LanguageIndex! = valor )
        {
            languageIndex = valor ;
            OnPropertyChanged ( "LanguageIndex" );

            Si (LanguageIndex> = 0 && languageIndex <languages.Length)
            {
                Language = idiomas [languageIndex];
                OnPropertyChanged ( "Idioma" );
            }
            TestIfValid ();
        }
    }
}
```

```

        }

        obtener { regreso languageIndex; }

    }

    public string Idioma { conjunto privado ; obtener ; }

    public bool Es válida
    {
        conjunto privado
        {
            Si (isValid!= valor)
            {
                isValid = valor ;
                OnPropertyChanged ("Es válida");
            }
        }
        obtener { regreso es válida; }
    }

    vacío TestIfValid ()
    {
        isValid != Cuerda .IsNullOrEmptySpace (Nombre) &&
        ! Cuerda .IsNullOrEmptySpace (Email) &&
        ! Cuerda .IsNullOrEmptySpace (Idioma);
    }

    vacío OnPropertyChanged ( cuerda nombre de la propiedad)
    {
        PropertyChangedEventHandler handler = PropertyChanged;

        Si (Manejador!= nulo )
            entrenador de animales( esta , nuevo PropertyChangedEventArgs (nombre de la propiedad));
    }
}
}
}

```

los Nombre y Email propiedades son de tipo cuerda con el propósito de unirse con el Texto propiedades de una Entrada elemento.
los LanguageIndex propiedad está destinado a ser unido a la Selecte-
dindex propiedad de la Recogedor. Pero el conjunto descriptor de acceso para LanguageIndex utiliza ese valor para ajustar el
Idioma propiedad de tipo cuerda de una matriz de cadenas en el idiomas colección.

Cada vez que el Nombre Correo Electronico, o LanguageIndex cambios de propiedad, la TestIfValid método se llama para establecer la Es válida propiedad. Esta propiedad se puede unir a la Está habilitado propiedad de la Botón.

La página de inicio en **MvvmEnforcement** es el mismo que el de **ModalEnforcement**, pero por supuesto el archivo XAML para la página modal es un poco más simple e implementa todos los enlaces de datos:

```

< Página de contenido xmlns = "http://xamarin.com/schemas/2014/forms"
    xmlns : X = "Http://schemas.microsoft.com/winfx/2009/xaml"
    X : Clase = "MvvmEnforcement.MvvmEnforcementModalPage"
    Título = "Modal Página">
```

```

< StackLayout Relleno = "20, 0">
    < Entrada Texto = "{ Unión Nombre }"
        marcador de posición = "Entrar nombre"
        VerticalOptions = "CenterAndExpand" />

    < Entrada Texto = "{ Unión Email }"
        marcador de posición = "Introducir dirección de correo electrónico"
        VerticalOptions = "CenterAndExpand" />

    < Recogedor X : Nombre = "Selector"
        Título = "Lenguaje de programación favorito"
        SelectedIndex = "{ Unión LanguageIndex }"
        VerticalOptions = "CenterAndExpand" />

    < Botón Texto = "Done"
        Está habilitado = "{ Unión Es válida }"
        HorizontalOptions = "Centro"
        VerticalOptions = "CenterAndExpand"
        hecho clic = "OnDoneButtonClicked" />

</ StackLayout >
</ Página de contenido >

```

El marcado contiene cuatro enlaces con propiedades en el modelo de vista.

archivo de código subyacente de la página modal es responsable de instanciar LittleViewModel y establecer el objeto de la BindingContext propiedad de la página, lo que lo hace en el constructor. El constructor también tiene acceso a la idiomas colección del modelo de vista para establecer el Artículos propiedad de la

Recogedor. (Desafortunadamente, el Artículos propiedad no está respaldado por una propiedad enlazable y por lo tanto no es enlazable.)

El resto del archivo es bastante similar a la página modal en **ModalEnforcement** excepto que el OnBackButtonPressed accede override la Es válida propiedad de LittleViewModel para determinar si debe llamar a la implementación de la clase base o de retorno cierto:

```

público clase parcial MvvmEnforcementModalPage : Página de contenido
{
    público MvvmEnforcementModalPage ()
    {
        InitializeComponent ();

        LittleViewModel viewmodel = nuevo LittleViewModel ();
        BindingContext = modelo de vista;

        // Llenar Selector de lista Elementos.
        para cada ( cuerta idioma en viewModel.Languages )
        {
            picker.Items.Add (idioma);
        }
    }

    protegido bool anulación OnBackButtonPressed ()
    {
        LittleViewModel ViewModel = ( LittleViewModel ) BindingContext;
    }
}

```

```
    regreso viewModel.IsValid? base.OnBackButtonPressed (): cierto ;
}

vacío asíncrono OnDoneButtonClicked ( objeto remitente, EventArgs args)
{
    esperar Navigation.PopModalAsync ();
}
}
```

variaciones de navegación

Como usted ha experimentado con el **ModalEnforcement** y **MvvmEnforcement** programas, es posible que se han sentido desconcertados por el fracaso de las páginas modales para retener ninguna información. Hemos todos los programas y sitios web que se encuentran navegar a una página utilizada para introducir información, pero al salir de esa página y luego regresar más tarde, toda la información que ha introducido se ha ido! Este tipo de páginas puede ser muy molesto.

Incluso en muestras simples como demostración **ModalEnforcement** y **MvvmEnforcement**, es posible solucionar ese problema muy fácilmente mediante la creación de una única instancia de la página modal, tal vez cuando el programa se pone en marcha, y luego usar esa instancia única en todo.

A pesar de la aparente facilidad de esta solución, no es un buen enfoque generalizado al problema de la retención de información de la página. Esta técnica probablemente se debe evitar a excepción de los casos más sencillos. Mantener una gran cantidad de páginas activas podría dar lugar a problemas de memoria, y hay que tener cuidado para no tener la misma instancia de la página en la pila de navegación más de una vez.

Sin embargo, aquí es cómo se puede modificar el **ModalEnforcementHomePage** archivo de código subyacente para esta técnica:

```
público clase parcial ModalEnforcementHomePage : Página de contenido
{
    ModalEnforcementModalPage modalPage = nuevo ModalEnforcementModalPage ();

    público ModalEnforcementHomePage ()
    {
        InitializeComponent ();
    }

    vacío asíncrono OnGoToButtonClicked ( objeto remitente, EventArgs args)
    {
        esperar Navigation.PushModalAsync (modalPage);
    }
}
```

los **ModalEnforcementHomePage** ahorra una instancia de **ModalEnforcementModalPage** como un campo y después se pasa siempre que sola instancia a **PushModalAsync**.

En aplicaciones menos simples, esta técnica puede salir mal: A veces, un tipo particular de la página

en una aplicación se puede navegar a partir de varias páginas diferentes, y que podría resultar en dos casos separados, inconsistentes de ModalPage.

Esta técnica se derrumba por completo si es necesario guardar el estado del programa cuando se termina y restaurarlo cuando se ejecuta de nuevo. No se puede guardar y restaurar las instancias página mismos. Por lo general los datos asociados con la página que hay que salvar.

En la programación de la vida real, ViewModels a menudo forman la columna vertebral de los tipos de páginas en una aplicación de varias páginas, y la mejor manera de que una aplicación puede retener datos de la página es a través del modelo de vista en lugar de la página.

Una mejor manera de mantener el estado de la página cuando una página modal se invoca varias veces de forma sucesiva se puede demostrar utilizando **MvvmEnforcement**. En primer lugar, añadir un alojamiento a la Aplicación página para LittleViewModel y una instancia de esa clase en el Aplicación constructor:

```
espacio de nombres MvvmEnforcement
{
    clase pública Aplicación : Solicitud
    {
        público App ()
        {
            ModalPageViewModel = nuevo LittleViewModel ();

            MainPage = nuevo NavigationPage ( nuevo MvvmEnforcementHomePage ());
        }

        público LittleViewModel ModalPageViewModel { conjunto privado ; obtener ; }
        ...
    }
}
```

Porque el LittleViewModel se instancia una sola vez, mantiene la información de la duración de la aplicación. Cada nueva instancia de MvvmEnforcementModalPage puede entonces simplemente acceder a este inmueble y establecer el ViewModel oponerse a su BindingContext:

```
público clase parcial MvvmEnforcementModalPage : Pagina de contenido
{
    público MvvmEnforcementModalPage ()
    {
        InitializeComponent ();

        LittleViewModel ViewModel = (( Aplicación ) Solicitud .Current) .ModalPageViewModel;
        BindingContext = modelo de vista;

        // Llenar Selector de lista Elementos.
        para cada ( cuerda idioma en viewModel.Languages)
        {
            picker.Items.Add (idioma);
        }
    }
    ...
}
```

Por supuesto, una vez que el programa termina, la información se pierde, pero el Aplicación clase también puede guardar esa información en el propiedades propiedad de Solicitud técnica -a demostró por primera vez en el PersistentKeypad programa hacia el final del capítulo 6, "clics de botón", y luego recuperarlo cuando la aplicación se inicia de nuevo.

Los problemas de retención de datos y pasar datos entre páginas-ocuparán gran parte del enfoque de las secciones posteriores de este capítulo.

Hacer un menú de navegación

Si su aplicación consiste en una variedad de diferentes pero arquitectónicamente idénticos páginas, todos los cuales son navegables desde la página principal, que podría estar interesado en la construcción de lo que se llama a veces

Menú de Navegación. Se trata de un menú en el que cada entrada es un tipo de página en particular.

los **ViewGalleryType** programa está destinado a demostrar todo el Ver clases en Xamarin.Forms. Contiene una página de inicio y una página para cada clase instanciable en Xamarin.Forms que se deriva de **Ver** -pero no **Diseño** -con la excepción de **Mapa** y **OpenGLView**. Eso es 18 clases y 18 **Esta-
tentPage** derivados, además de la página principal. (La razón de la **Tipo** sufijo en el nombre del proyecto se pondrá de manifiesto en breve).

Estos 18 clases de página se almacenan en una carpeta denominada **ViewPages** en la biblioteca de clases portátil. Aquí está un ejemplo: **SliderPage.xaml**. Es solamente un deslizador con un Etiqueta unido a la Valor propiedad:

```
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " ViewGalleryType.SliderPage "
    Título = " deslizador " >

    < StackLayout Relleno = " 10, 0 " >
        < deslizador x: Nombre = " deslizador "
            VerticalOptions = " CenterAndExpand " >

            < Etiqueta Texto = " {Binding Fuente = {x: deslizador de referencia}},
                Path = Valor,
                StringFormat = "El valor del deslizador es {0} " "
                VerticalOptions = " CenterAndExpand "
                Alineación horizontal = " Centro " />

        </ StackLayout >
    </ Pagina de contenido >
```

El otro 17 son similares. Algunas de las páginas tienen un poco de código en el archivo de código subyacente, pero la mayoría de ellos simplemente tienen una llamada a **InitializeComponent**.

además, el **ViewGalleryType** proyecto tiene una carpeta con el nombre **imágenes** que contiene 18 mapas de bits con el nombre de cada **Ver** derivado extendió hasta casi llenar la superficie del mapa de bits. Estos mapas de bits fueron generados por un programa de Windows Presentation Foundation y se marcan como **EmbeddedResource** en el proyecto. El proyecto contiene también una **ImageResourceExtension** la clase descrita en el capítulo 13, en la sección "recursos incrustados," para hacer referencia a los mapas de bits del archivo XAML.

La página principal se llama ViewGalleryTypePage. Reúne 18 ImageCell elementos en seis diferentes secciones de una Mesa:

```
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    xmlns: locales = "clr-espacio de nombres: ViewGalleryType; montaje = ViewGalleryType "
    x: Class = "ViewGalleryType.ViewGalleryTypePage "
    Titulo = "Ver galería "
    < TableView Intención = "Menú " >
        < TableRoot >
            < TableSection Título = "Vistas de presentación " >
                < ImageCell Fuente de imagen = "{Locales: ImageResource ViewGalleryType.Images.Label.png}"
                    Texto = "Texto de la pantalla"
                    Mando = "(Binding) NavigateCommand"
                    CommandParameter = "{X: Local Tipo: LabelPage}" />

                < ImageCell Fuente de imagen = "{Locales: ImageResource ViewGalleryType.Images.Image.png}"
                    Texto = "Mostrar un mapa de bits"
                    Mando = "(Binding) NavigateCommand"
                    CommandParameter = "{X: Local Tipo: ImagePage}" />

                < ImageCell Fuente de imagen =
                    "{Locales: ImageResource ViewGalleryType.Images.BoxView.png}"
                    Texto = "Mostrar un bloque"
                    Mando = "(Binding) NavigateCommand"
                    CommandParameter = "{X: Local Tipo: BoxViewPage}" />

                < ImageCell Fuente de imagen =
                    "{Locales: ImageResource ViewGalleryType.Images.WebView.png}"
                    Texto = "Mostrar una página web"
                    Mando = "(Binding) NavigateCommand"
                    CommandParameter = "{X: Local Tipo: WebViewPage}" />
            </ TableSection >

            < TableSection Título = "Vistas de mando " >
                < ImageCell Fuente de imagen = "{Locales: ImageResource ViewGalleryType.Images.Button.png}"
                    Texto = "Iniciar el mando"
                    Mando = "(Binding) NavigateCommand"
                    CommandParameter = "{X: Local Tipo: ButtonPage}" />

                < ImageCell Fuente de imagen =
                    "{Locales: ImageResource ViewGalleryType.Images.SearchBar.png}"
                    Texto = "Iniciar una búsqueda de texto"
                    Mando = "(Binding) NavigateCommand"
                    CommandParameter = "{X: Local Tipo: SearchBarPage}" />
            </ TableSection >

            < TableSection Título = "Datos de tipo Vistas " >
                < ImageCell Fuente de imagen = "{Locales: ImageResource ViewGalleryType.Images.Slider.png}"
                    Texto = "Rango de dobles"
                    Mando = "(Binding) NavigateCommand"
                    CommandParameter = "{X: Local Tipo: SliderPage}" />

                < ImageCell Fuente de imagen =
```

```
" (Locales: ImageResource ViewGalleryType.Images.Stepper.png) "
Texto = " dobles discretos "
Mando = " {Binding} NavigateCommand "
CommandParameter = " {X: Local Tipo: StepperPage} " />

<ImageCell Fuente de imagen = " {Locales: ImageResource ViewGalleryType.Images.Switch.png} "
Texto = " Seleccionar verdadero o falso "
Mando = " {Binding} NavigateCommand "
CommandParameter = " {X: Local Tipo: SwitchPage} " />

<ImageCell Fuente de imagen =
" (Locales: ImageResource ViewGalleryType.Images.DatePicker.png) "
Texto = " Seleccione una fecha "
Mando = " {Binding} NavigateCommand "
CommandParameter = " {X: Local Tipo: DatePickerPage} " />

<ImageCell Fuente de imagen =
" (Locales: ImageResource ViewGalleryType.Images.TimePicker.png) "
Texto = " Seleccione una hora "
Mando = " {Binding} NavigateCommand "
CommandParameter = " {X: Local Tipo: TimePickerPage} " />

</TableSection >

<TableSection Título = " Vistas de edición de texto " >
<ImageCell Fuente de imagen = " {Locales: ImageResource ViewGalleryType.Images.Entry.png} "
Texto = " Edición de una sola línea "
Mando = " {Binding} NavigateCommand "
CommandParameter = " {X: Local Tipo: EntryPage} " />

<ImageCell Fuente de imagen = " {Locales: ImageResource ViewGalleryType.Images.Editor.png} "
Texto = " Editar un párrafo "
Mando = " {Binding} NavigateCommand "
CommandParameter = " {X: Local Tipo: EditorPage} " />

</TableSection >

<TableSection Título = " El indicador de actividad Vistas " >
<ImageCell Fuente de imagen =
" (Locales: ImageResource ViewGalleryType.Images.ActivityIndicator.png) "
Texto = " Mostrar la actividad "
Mando = " {Binding} NavigateCommand "
CommandParameter = " {X: Local Tipo: ActivityIndicatorPage} " />

<ImageCell Fuente de imagen =
" (Locales: ImageResource ViewGalleryType.Images.ProgressBar.png) "
Texto = " mostrar el progreso "
Mando = " {Binding} NavigateCommand "
CommandParameter = " {X: Local Tipo: ProgressBarPage} " />

</TableSection >

<TableSection Título = " Vistas Collection " >
<ImageCell Fuente de imagen = " {Locales: ImageResource ViewGalleryType.Images.Picker.png} "
Texto = " Recoger elemento de la lista "
Mando = " {Binding} NavigateCommand "
CommandParameter = " {X: Local Tipo: PickerPage} " />
```

```

< ImageCell Fuente de imagen =
    " {Locales: ImageResource ViewGalleryType.Images.ListView.png} "
    Texto = " Mostrar una colección "
    Mando = " {Binding} NavigateCommand "
    CommandParameter = " {x: Local Tipo: ListViewPage} " />

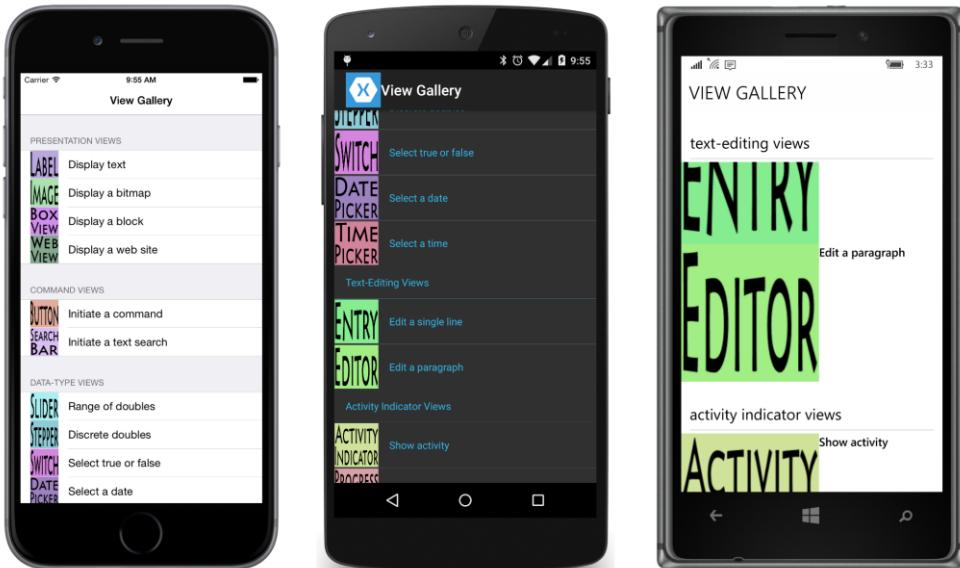
< ImageCell Fuente de imagen =
    " {Locales: ImageResource ViewGalleryType.Images.TableView.png} "
    Texto = " Mostrar formulario o en el menú "
    Mando = " {Binding} NavigateCommand "
    CommandParameter = " {x: Local Tipo: TableViewPage} " />

</ TableSection >
</ TableRoot >
</ TableView >
</ Página de contenido >

```

Cada ImageCell tiene una referencia a un mapa de bits que indica el nombre y una de las vistas Texto propiedad que se describe brevemente la vista. los Mando propiedad de ImageCell está unido a un Yo ordeno objeto que se implementa en el archivo de código subyacente, y la CommandParameter es una x: Tipo extensión de marcado que hace referencia a una de las clases de página. Como se recordará, el x: Tipo extensión de marcado es el XAML equivalente de la C # tipo de operador y resultados en cada CommandParameter siendo de tipo Tipo.

Esto es lo que la página de inicio se ve como en las tres plataformas:



El archivo de código subyacente define el NavigateCommand propiedad de que cada ImageCell referencias en una unión. los Ejecutar método se implementa como una función lambda: Se pasa el Tipo argumento (conjunto de la CommandParameter en el archivo XAML) a Activator.CreateInstance crear una instancia de la página y luego navega a la página:

```

pública clase parcial ViewGalleryTypePage : Pagina de contenido
{
    pública ViewGalleryTypePage ()
    {
        InitializeComponent ();

        NavigateCommand = nuevo Mando < Tipo > ( asincrono ( Tipo quiénes somos? ) =>
        {
            Página page = ( Página ) Activador.CreateInstance ( quiénes somos? );
            esperar Navigation.PushAsync ( página );
        });
    }

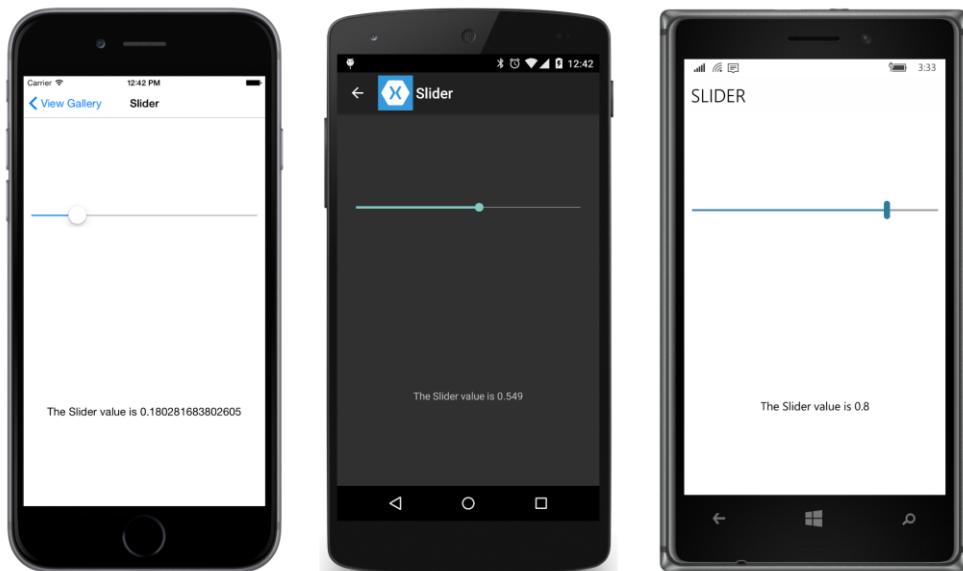
    BindingContext = esta ;
}

pública Yo ordeno NavigateCommand { conjunto privado ; obtener ; }
}

```

El constructor concluye estableciendo su BindingContext propiedad a sí mismo, por lo que cada ImageCell en el archivo XAML puede hacer referencia a la NavigateCommand propiedad con un simple Unión.

grabar el deslizador entrada (por ejemplo) se desplaza a SliderPage:



Volviendo a la página principal requiere el uso de la barra de navegación en las pantallas de iOS y Android o el **Espalda** botón en las pantallas móviles Android y Windows 10.

Una nueva instancia de cada página se crea cada vez que navega a esa página, así que por supuesto estas diferentes instancias de SliderPage no retendrá el valor de la deslizador es posible que haya establecido previamente.

¿Es posible crear una sola instancia de cada una de estas 18 páginas? Sí, y eso se demuestra en **ViewGalleryInst**. Los **Inst** sufijo significa “ejemplo” para distinguir el programa de la utilización de un tipo de página en **ViewGalleryType**.

Las 18 clases de página para cada vista son los mismos, como son los mapas de bits. La página principal, sin embargo, ahora expresa la **CommandParameter** propiedad de cada **ImageCell** como un elemento de propiedad para crear instancias de cada clase de página. He aquí un extracto:

```
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    xmlns: locales = " dir-espacio de nombres: ViewGalleryInst; montaje = ViewGalleryInst "
    x: Class = " ViewGalleryInst.ViewGalleryInstPage "
    Titulo = " Ver galería " >

< TableView Intención = " Menú " >
    < TableRoot >
        < TableSection Titulo = " Vistas de presentación " >
            < ImageCell Fuente de imagen = " {Locales: ImageResource ViewGalleryInst.Images.Label.png} "
                Texto = " texto de la pantalla "
                Mando = " {Binding} NavigateCommand " >
                < ImageCell.CommandParameter >
                    < locales: LabelPage />
                </ ImageCell.CommandParameter >
            </ ImageCell >
            ...
            < ImageCell Fuente de imagen =
                " {Locales: ImageResource ViewGalleryInst.Images.TableView.png} "
                Texto = " Mostrar formulario o en el menú "
                Mando = " {Binding} NavigateCommand " >
                < ImageCell.CommandParameter >
                    < locales: TableViewPage />
                </ ImageCell.CommandParameter >
            </ ImageCell >
        </ TableSection >
    </ TableRoot >
</ TableView >
</ Pagina de contenido >
```

Ahora bien, cuando se manipula la deslizadora sobre el **SliderPage**, y luego volver a casa y vaya a la **SliderPage** de nuevo, el deslizador será la misma porque es la misma instancia de página.

Tenga en cuenta que con esta configuración, se crea una instancia de un total de 19 clases de página cuando el programa se pone en marcha, y eso significa que 19 archivos XAML están siendo analizados, y que podría afectar el rendimiento de inicio, y ocupar una gran cantidad de memoria también.

Por otra parte, cualquier error en los archivos XAML que se encuentran durante este tiempo de ejecución de análisis también se manifestarán al inicio del programa. Podría ser difícil de descubrir exactamente qué archivo XAML tiene el problema! Cuando la construcción de un programa que crea la instancia de muchas clases de página en una sola toma, tendrá que añadir nuevas clases de forma incremental para asegurarse de que todo funciona bien antes de continuar.

Lo mejor para evitar esta técnica en su totalidad. Crear una instancia de cada página a medida que lo necesite, y conservar los datos asociados a la página mediante el uso de un modelo de vista.

La manipulación de la pila de navegación

A veces es necesario alterar el flujo normal orientado a pila de navegación. Por ejemplo, supongamos que una página necesita un poco de información del usuario, pero primero navega a una página que proporciona algunas instrucciones o un descargo de responsabilidad, y luego a partir de ahí se desplaza a la página que en realidad obtiene la información. Cuando el usuario ha terminado y se remonta, tendrá que saltarse esa página con las instrucciones o descargo de responsabilidad. Esa página debe ser retirada de la pila de navegación.

Aquí está un ejemplo similar: Supongamos que el usuario está interactuando con una página que obtiene alguna información y luego quiere volver a la página anterior. Sin embargo, el programa detecta que algo está mal con esta información que requiere una extensa discusión en una página separada. El programa podría insertar una nueva página en la pila de navegación para proporcionar esa discusión.

O una cierta secuencia de páginas podría terminar con una Botón etiquetado Ir a casa, y todas las páginas en el medio, simplemente se pueden saltar cuando se navega de nuevo a la página principal.

los INavigation interfaz define métodos para los tres de estos casos. Se denominan Retirar-Página, InsertPageBefore, y PopToRootAsync.

los StackManipulation programa demuestra estos tres métodos, pero de una manera muy abstracta. El programa consta de cinco páginas de códigos únicos, llamado PáginaA, página B, PageBAlternative, PageC, y Paginado. Cada página establece su Título propiedad para identificarse.

PáginaA tiene un Botón para desplazarse hasta Página B:

```
clase pública páginaA : Pagina de contenido
{
    público PáginaA ()
    {
        Botón botón = nuevo Botón
        {
            text = "Ir a la página B",
            Tamaño de Letra = Dispositivo .GetNamedSize ( NamedSize .Grande, tipo de ( Botón )),
            HorizontalOptions = LayoutOptions .Centrar,
            VerticalOptions = LayoutOptions .Centrar
        };
        button.Clicked += asíncrono ( Remitente, args ) =>
        {
            esperar Navigation.PushAsync ( nuevo Página B ());
        };
        title = "Página A";
        content = nuevo botón;
    }
}
```

Página B es similar, excepto que se desplaza a PageC. PageBAlternative es lo mismo que Página B excepto que se identifica como "Página B Alt". PageC tiene un Botón para desplazarse hasta paginado, y paginado tiene dos botones:

```
clase pública paginado : Pagina de contenido
```

```
{  
    público Paginada ()  
    {  
        // Crear botón para ir directamente a la página A.  
        Botón homeButton = nuevo Botón  
        {  
            text = "Ir directamente a casa",  
            Tamaño de Letra = Dispositivo .GetNamedSize ( NamedSize .Grande, tipo de ( Botón )),  
            HorizontalOptions = LayoutOptions .Centrar,  
            VerticalOptions = LayoutOptions .CenterAndExpand  
        };  
  
        homeButton.Clicked += asíncrono (Remitente, args) =>  
        {  
            esperar Navigation.PopToRootAsync ();  
        };  
  
        // Crear botón para intercambiar páginas.  
        Botón swapButton = nuevo Botón  
        {  
            text = "Swap B y Alt B",  
            Tamaño de Letra = Dispositivo .GetNamedSize ( NamedSize .Grande, tipo de ( Botón )),  
            HorizontalOptions = LayoutOptions .Centrar,  
            VerticalOptions = LayoutOptions .CenterAndExpand  
        };  
  
        swapButton.Clicked += (remitente, args) =>  
        {  
            IReadOnlyList < Página > NavStack = Navigation.NavigationStack;  
            Página pageC = navStack [navStack.Count - 2];  
            Página existingPageB = navStack [navStack.Count - 3];  
            bool isOriginal = existingPageB es Página B ;  
            Página newPasswordB = isOriginal? ( Página ) nuevo PageBAlternative () : nuevo Página B ();  
  
            // Intercambiar las páginas.  
            Navigation.RemovePage (existingPageB);  
            Navigation.InsertPageBefore (newPageB, pageC);  
  
            // Acabado: deshabilitar el botón.  
            swapButton.IsEnabled = falso ;  
        };  
  
        title = "Página D";  
        content = nuevo StackLayout  
        {  
            Los niños =  
            {  
                botón de inicio,  
                swapButton  
            }  
        };  
    };  
}
```

El botón etiquetado **Ir directamente a la casa** tiene un hecho clic controlador que llama PopToRootAsync. Esto hace que el programa para saltar de nuevo a páginaA y borra efectivamente la pila de navegación de todas las páginas intermedias.

El botón etiquetado **Intercambiar B y Alt B** es un poco más compleja. Los hecho clic manejador para este botón sustituye Página B con PageBAlternative en la pila de navegación (o viceversa), por lo que al volver a través de las páginas, se encontrará con una página diferente B. He aquí cómo el hecho clic manejador lo hace:

En el momento de la Botón se hace clic, el NavigationStack tiene cuatro elementos con índices 0, 1, 2, y 3. Estos cuatro índices corresponden a los objetos en la pila de tipo PáginaA, página B (o PageBAlternative),

PageC, y Paginado. accede al controlador de la NavigationStack para obtener estos casos reales:

```
IReadOnlyList < Página > NavStack = Navigation.NavigationStack;
Página pageC = navStack [navStack.Count - 2];
Página existingPageB = navStack [navStack.Count - 3];
```

Ese existingPageB objeto podría ser de tipo Página B o PageBAlternative, por lo que una newPageB objeto se crea del otro tipo:

```
bool isOriginal = existingPageB es Página B ;
Página newPageB = isOriginal? ( Página ) nuevo PageBAlternative (): nuevo Página B ;
```

Las dos afirmaciones siguientes quitan el existingPageB objeto de la pila de navegación y luego insertar el newPageB objeto en la ranura antes de pageC, intercambiando con eficacia las páginas:

```
// Intercambiar las páginas.
Navigation.RemovePage (existingPageB);
Navigation.InsertPageBefore (newPageB, pageC);
```

Obviamente, la primera vez que hace clic en este botón, existingPageB será una Página B objeto y nuevo-Página B será una PageBAlternative objeto, pero a continuación, puede volver a PageC o PageBAlternative, y navegar hacia delante de nuevo para Paginado. Al hacer clic en el botón de nuevo sustituirá al PageBAlternative objeto con una Página B objeto.

generación de páginas dinámicas

los **BuildAPage** programa es una aplicación de varias páginas, pero el **BuildAPage** proyecto contiene sólo una única clase de página llamada **BuildAPageHomePage**. Como su nombre indica, el programa construye una nueva página de código y luego se desplaza a la misma.

El archivo XAML permite especificar lo que quiere en esta página construida:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " BuildAPage.BuildAPageHomePage "
    Título = " Build-a-Página "
    Relleno = " 10, 5 " >

< StackLayout >
    < Etiqueta Texto = " Ingrese título de la página: " />
```

```

< Entrada x: Nombre = "titleEntry "
    marcador de posición = " Título de la página " />

< Cuadrícula VerticalOptions = " FillAndExpand " >
    < ContentView Grid.Row = " 0 " >
        < StackLayout >
            < Etiqueta Texto = " Toque para agregar a la página generada: " />
            < Vista de la lista x: Nombre = " ver lista "
                itemSelected = " OnViewListItemSelected " >
                < ListView.ItemsSource >
                    < x: Array Tipo = " (X: x.Tipo: String) " >
                        < x: String > BoxView </x: String >
                        < x: String > Botón </x: String >
                        < x: String > Selector de fechas </x: String >
                        < x: String > Entrada </x: String >
                        < x: String > deslizador </x: String >
                        < x: String > paso a paso </x: String >
                        < x: String > Cambiar </x: String >
                        < x: String > TimePicker </x: String >
                    </x: Array >
                </ ListView.ItemsSource >
            </ Vista de la lista >
        </ StackLayout >
    </ ContentView >

    < ContentView Grid.Row = " 1 " >
        < StackLayout >
            < Etiqueta Texto = " Toque para eliminar de la página generada: " />
            < Vista de la lista x: Nombre = " PageList "
                itemSelected = " OnPageListItemSelected " >
        </ StackLayout >
    </ ContentView >
</ Cuadrícula >

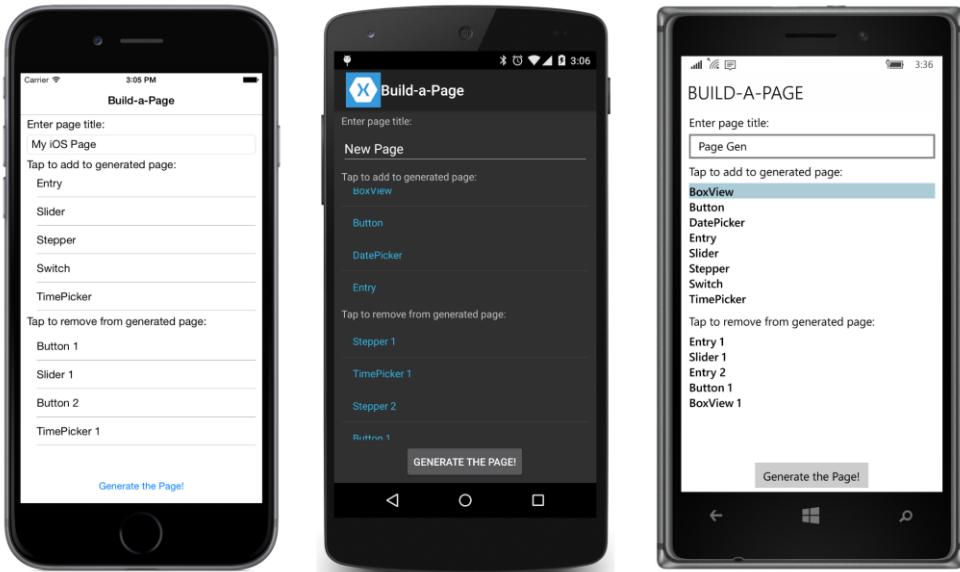
< Botón x: Nombre = " generateButton "
    Texto = " Generación de la página! "
    Está habilitado = " Falso "
    HorizontalOptions = " Centrar "
    VerticalOptions = " Centrar "
    hecho clic = " OnGenerateButtonClicked " />
</ StackLayout >
</ Página de contenido >

```

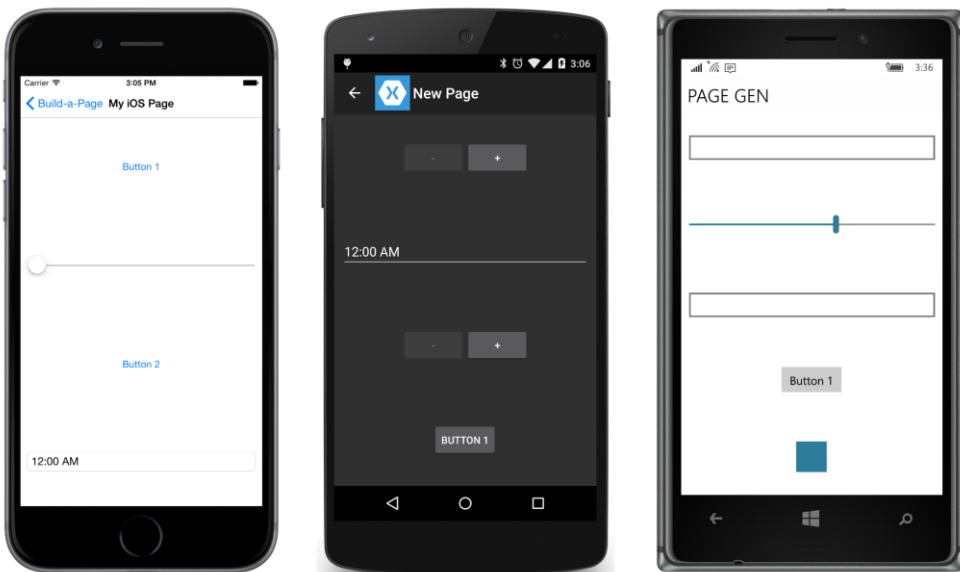
Utilizar el Entrada en la parte superior de la página para especificar una Título Propiedad en la página construida. UN

Vista de la lista a continuación, enumera ocho puntos de vista comunes que puede que desee en su página. A medida que selecciona estos puntos de vista, se transfieren a la segunda Vista de la lista. Si desea eliminar uno de ellos de la página, simplemente toque en esta segunda Vista de la lista.

Así es como podría parecer después de haber seleccionado una serie de elementos de la página. Observe que puede seleccionar varios elementos del mismo tipo y cada uno se le asigna un número único:



Cuando haya terminado de todos “diseñar” su página, simplemente pulse el **Generación de la página!** botón en la parte inferior, y el programa se basa esa página y se desplaza a la misma:



El archivo de código subyacente tiene `itemSelected` manejadores para los dos Vista de la lista elementos para agregar elementos y eliminar elementos de la segunda Vista de la lista, pero el procesamiento más interesante ocurre en el hecho clic controlador para el Botón:

```
público clase parcial BuildAPageHomePage : Pagina de contenido
```

```
{  
    ObservableCollection<cuenda> = ViewCollection nuevo ObservableCollection<cuenda>();  
    Montaje xamarinForms = tipo de (Etiqueta) .GetTypeInfo () Asamblea.;  
  
    público BuildAPageHomePage ()  
    {  
        InitializeComponent ();  
  
        pageList.ItemsSource = viewCollection;  
    }  
  
    vacío OnViewListItemSelected ( objeto remitente, SelectedItemChangedEventArgs args)  
    {  
        Si (Args.SelectedItem = nulo )  
        {  
            viewList.SelectedItem = nulo ;  
            En t número = 1;  
            cuenda item = nulo ;  
  
            mientras (-1! = ViewCollection.IndexOf (  
                item = ((cuenda ) Args.SelectedItem) + " + Número))  
            {  
                número ++;  
            }  
  
            viewCollection.Add (elemento);  
            generateButton.IsEnabled = cierto ;  
        }  
    }  
  
    vacío OnPageListItemSelected ( objeto remitente, SelectedItemChangedEventArgs args)  
    {  
        Si (Args.SelectedItem = nulo )  
        {  
            pageList.SelectedItem = nulo ;  
            viewCollection.Remove ((cuenda ) Args.SelectedItem);  
            generateButton.IsEnabled = viewCollection.Count> 0;  
        }  
    }  
  
    vacío asíncrono OnGenerateButtonClicked ( objeto remitente, EventArgs args)  
    {  
        Pagina de contenido ContentPage = nuevo Pagina de contenido  
        {  
            Title = titleEntry.Text,  
            Relleno = nuevo Espesor (10, 0)  
        };  
        StackLayout stackLayout = nuevo StackLayout (0);  
        contentPage.Content = stackLayout;  
  
        para cada ( cuenda it en viewCollection)  
        {  
            cuenda viewBoxString = item.Substring (0, item.IndexOf ( " ));  
            Tipo viewType = xamarinForms.GetType ("Xamarin.Forms" .+ viewBoxString);  
        }  
    }  
}
```

```

Ver view = ( Ver ) Activador.CreateInstance (ViewType);
view.VerticalOptions = LayoutOptions.CenterAndExpand;

cambiar (ViewString)
{
    caso "BoxView" :
        (( BoxView ) Vista).Color = Color.Acento;
    caso Goto "Paso a paso" ;

    caso "Botón" :
        (( Botón ) Vista).Texto = artículo;
    caso Goto "Paso a paso" ;

    caso "Paso a paso" ;
    caso "Cambiar" :
        view.HorizontalOptions = LayoutOptions.Centrar;
        descanso ;
}
stackLayout.Children.Add (vista);
}
esperar Navigation.PushAsync (ContentPage);
}
}
}

```

Esta hecho clic crea un manejador Pagina de contenido y una StackLayout y luego simplemente bucles a través de las cuerdas en la segunda Vista de la lista, la búsqueda de una correspondiente Tipo objeto mediante el GetType

método definido por la Montaje clase. (Observe la Montaje objeto nombrado xamarinForms se define como un campo.) Una llamada a Activator.CreateInstance crea el elemento real, que luego se puede adaptar ligeramente para el diseño final.

Creando un Pagina de contenido objeto en el código y añadir elementos a que no es nuevo. De hecho, la plantilla de proyecto Xamarin.Forms estándar incluye una Aplicación clase con un constructor que crea una instancia Contenido-Página y añade una Etiqueta a la misma, por lo que se introdujeron a esta técnica camino de regreso en el capítulo 2. Sin embargo, este enfoque fue abandonado rápidamente en favor de la técnica más flexible de derivar una clase de Estafa-tentPage. Derivación de una clase es más potente porque la clase derivada tiene acceso a los métodos protegidas como OnAppearing y OnDisappearing.

A veces, sin embargo, es útil volver a lo básico.

Los patrones de transferencia de datos

A menudo es necesario para las páginas de una aplicación de varias páginas para compartir datos, y en particular para una página para pasar información a otra página. A veces este proceso se asemeja a una llamada de función: Cuando

Página principal muestra una lista de artículos y navega hasta detalle para mostrar una vista detallada de uno de estos elementos, Página principal debe pasar a ese tema en particular De detalle. O cuando el usuario introduce la información en FillOutFormPage, que la información debe ser devuelto de nuevo a la página que invoca FillOutFormPage.

Varias técnicas están disponibles para transferir datos entre páginas. La que utilice depende de la aplicación particular. Tenga en cuenta en toda esta discusión que es probable que también es necesario para guardar el contenido de la página cuando la aplicación termina, y recuperar el contenido cuando el programa se pone en marcha de nuevo. Algunas de las técnicas de intercambio de datos son más propicias para guardar y restaurar el estado página que otros. Este problema se explora con más detalle más adelante en este capítulo.

argumentos de constructor

Cuando una página se desplaza a otra página y tiene que pasar los datos a esa página, una forma obvia de pasar esos datos es a través del constructor de la segunda página.

los **SchoolAndStudents** programa ilustra esta técnica. El programa hace uso de la **SchoolOfFineArt** biblioteca introducido en el Capítulo 19, "vistas Collection". El programa consta de dos páginas denominadas **SchoolPage** y **StudentPage**. Los **SchoolPage** clase utiliza una Vista de la lista para mostrar una lista desplegable de todos los estudiantes de la escuela. Cuando el usuario selecciona uno, el programa se desplaza a un **StudentdentPage** que muestra detalles acerca del estudiante individual. El programa es similar a la **SelectedStudentDetail** programa en el Capítulo 19, excepto que la lista y los detalles se han separado en dos páginas.

A continuación se SchoolPage. Para mantener las cosas lo más simple posible, la Vista de la lista utiliza una ImageCell para mostrar cada estudiante en la escuela:

```
< Pagina de contenido xmlns = "http://xamarin.com/schemas/2014/forms"
    xmlns : X = "Http://schemas.microsoft.com/winfx/2009/xaml"
    X : Clase = "SchoolAndStudents.SchoolPage"
    Título = "Escuela">

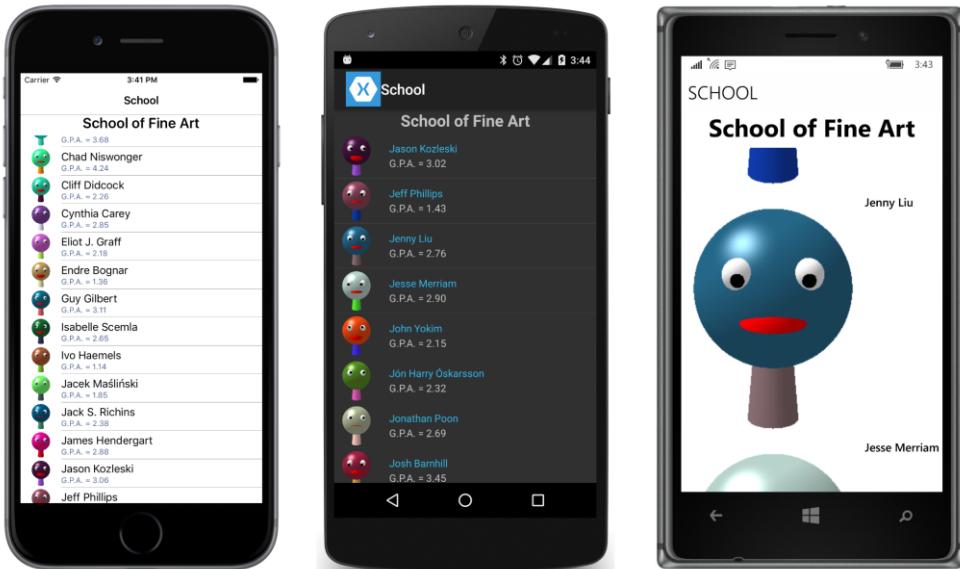
    < StackLayout BindingContext = "{ Unión Cuerpo de estudiantes }" >
        < Etiqueta Texto = "{ Unión Colegio }"
            Tamaño de fuente = "Large"
            FontAttributes = "Bold"
            HorizontalTextAlignment = "Center" />

        < Vista de la lista X : Nombre = "ListView"
            ItemsSource = "{ Unión estudiantes }"
            selectedItem = "OnListViewItemSelected">
            < ListView.ItemTemplate >
                < DataTemplate >
                    < ImageCell Fuente de imagen = "{ Unión PhotoFilename }"
                        Texto = "{ Unión Nombre completo }"
                        Detalle = "{ Unión Promedio de calificaciones ,
                            StringFormat = 'GPA = {0: F2}' }"/>
                </ DataTemplate >
            </ ListView.ItemTemplate >
        </ Vista de la lista >
    </ StackLayout >
</ Pagina de contenido >
```

Los enlaces de datos en este archivo XAML asumen que el BindingContext para la página se establece en un objeto de tipo **SchoolViewModel** se define en la **SchoolOfFineArt** Biblioteca. Los **SchoolViewModel** tiene un

propiedad de tipo Cuerpo de estudiantes, que se fija para el BindingContext del StackLayout. Los Etiquetas están unidos a la Colegio propiedad de Cuerpo de estudiantes, y el ItemsSource del Vista de la lista está unido a la estudiantes propiedad de colección de Cuerpo de estudiantes. Esto significa que cada elemento de la Vista de la lista tiene un BindingContext de tipo Estudiante. Los ImageCell hacen referencia a la PhotoFilename, NombreCompleto, y Promedio de calificaciones propiedades de ese Estudiante objeto.

He aquí que Vista de la lista que se ejecuta en iOS, Android y Windows Mobile 10:



El constructor en el archivo de código subyacente es responsable de establecer la BindingContext La página de una instancia de SchoolViewModel. El archivo de código subyacente también contiene un controlador para el ItemSelected evento:

```

pública clase parcial SchoolPage : Pagina de contenido
{
    pública SchoolPage ()
    {
        InitializeComponent ();

        // Conjunto BindingContext.
        BindingContext = nuevo SchoolViewModel ();
    }

    vacío asincrono OnListViewItemSelected ( objeto remitente, SelectedItemChangedEventArgs args)
    {
        // El elemento seleccionado es nulo o del tipo de estudiante.
        Estudiante estudiante = args.SelectedItem como Estudiante ;

        // Asegúrese de que un elemento se selecciona realmente.
        Si (Estudiante != nulo )
    }
}

```

```
// Desactive el artículo.  
listView.SelectedItem = null;  
  
// Vaya a StudentPage con el argumento del Estudiante.  
esperar Navigation.PushAsync (nuevo StudentPage (estudiante));  
}  
}
```

los Item seleccionado propiedad de los argumentos del evento es la Estudiante objeto que se toca, y el manejador lo usa como un argumento a la StudentPage clase en el PushAsync llamada.

Observe también que el controlador establece el `Item seleccionado` propiedad de la Vista de la lista a nulo. Esto anula la selección del tema para que no permanezca seleccionada cuando el usuario vuelve a la SchoolPage, y el usuario puede aprovechar de nuevo. Pero dejando que `Item seleccionado` propiedad a nulo También hace que otra llamada a la `itemSelected` controlador de eventos. Afortunadamente, el controlador ignora el evento si la `Item seleccionado` es nulo.

El archivo de código subyacente para StudentPage Simplemente utiliza ese argumento del constructor para establecer el EnlazamientoContexto de la página:

```
p\xf3blico clase parcial StudentPage : Pagina de contenido
{
    p\xf3blico StudentPage ( Estudiante estudiante)
    {
        InitializeComponent ();
        BindingContext = estudiante;
    }
}
```

El archivo XAML para el StudentPage clase contiene enlaces con varias propiedades de la Estudiante

```
< Pagina de contenido xmlns = "http://xamarin.com/schemas/2014/forms"
    xmlns : X = "Http://schemas.microsoft.com/winfx/2009/xaml"
    X : Clase = "SchoolAndStudents.StudentPage"
    Título = "Estudiante">

    < StackLayout >
        <! - Nombre ->
        < StackLayout Orientación = "Horizontal"
            HorizontalOptions = "Centro"
            Espaciado = "0">
            < StackLayout.Resources >
                < ResourceDictionary >
                    < Estilo Tipo de objetivo = "Etiqueta">
                        < Setter Propiedad = "FontSize" Valor = "Grande" />
                        < Setter Propiedad = "FontAttributes" Valor = "Negrita" />
                    </ Estilo >
                </ ResourceDictionary >
            </ StackLayout.Resources >
            < Etiqueta Texto = "Unión Apellido" />
```

```

<! - Nombre de pila ->
<! - Segundo nombre ->
</StackLayout>

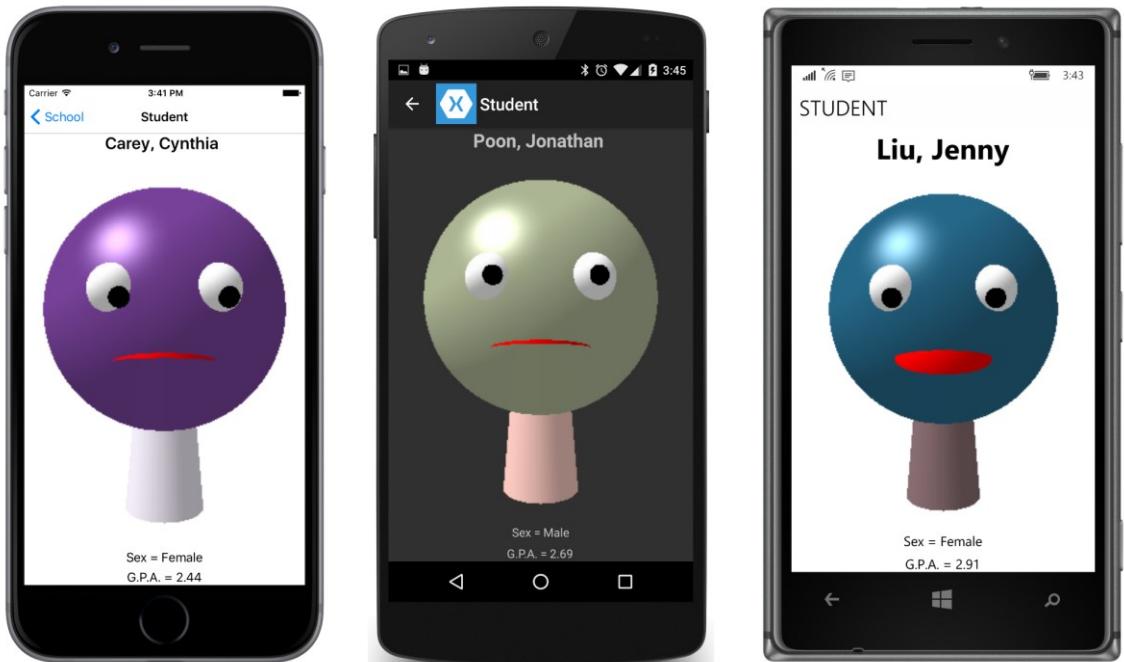
<! - Foto ->
<Imagen Fuente = "{ Unión PhotoFilename }"
VerticalOptions = "FillAndExpand" />

<! - Sexo ->
<Etiqueta Texto = "{ Unión Sexo , StringFormat = 'Sexo = {0}' }"
HorizontalOptions = "Center" />

<! - GPA ->
<Etiqueta Texto = "{ Unión Promedio de calificaciones , StringFormat = 'GPA = {0: F2}' }"
HorizontalOptions = "Center" />
</StackLayout>
</Pagina de contenido >

```

El archivo XAML no requiere un botón o cualquier otro objeto de interfaz de usuario para volver de nuevo a SchoolPage debido a que se proporciona de forma automática, ya sea como parte de la interfaz de usuario estándar de navegación de la plataforma o como parte del propio teléfono:



Pasar información a la página navegado a través del constructor es versátil, pero para este ejemplo en particular, es innecesaria. **StudentPage** podría tener un constructor sin parámetros, y el Colegio-Página podría establecer el **BindingContext** de la recién creada **StudentPage** justo en el **PushAsync** llamada:

```
esperar Navigation.PushAsync (nuevo StudentPage () BindingContext = estudiante);
```

Uno de los problemas con cualquiera de estos enfoques es preservar el estado de la aplicación cuando se suspende el programa. Si tu **quieres StudentPage para guardar el estudiante actual cuando el programa termina, tiene que guardar todas las propiedades de la Estudiante objeto.** Pero cuando esa Estudiante objeto se vuelve a crear cuando el programa se pone en marcha de nuevo, es un objeto diferente de lo particular Estudiante Objeto para el mismo estudiante en el estudiantes recogida a pesar de que todas las propiedades son las mismas.

Si el estudiantes colección es conocido por ser constante, que tiene más sentido para StudentPage para guardar sólo un índice en la estudiantes colección que hace referencia a este particular Estudiante objeto. Sin embargo, en este ejemplo, StudentPage no tiene acceso a ese índice o al estudiantes colección.

Propiedades y las llamadas a métodos

Un llamado página PushAsync o PushModalAsync obviamente, tiene acceso directo a la clase que se está navegando a, por lo que puede establecer las propiedades o llamar a los métodos en que objeto de página para pasar información a la misma. Un llamado página PopAsync o PopModalAsync, sin embargo, tiene más trabajo que hacer para determinar la página que se está volviendo a. En el caso general, una página no siempre se puede esperar que esté familiarizado con el tipo de página de la página que navegar a la misma.

Tendrá que tener cuidado al establecer las propiedades o llamar a los métodos de una página a otra. No se puede hacer ninguna suposición acerca de la secuencia de llamadas a la OnAppearing y

OnDisappearing anulaciones y la finalización de la PushAsync, PopAsync, PushModalAsync, y
PopModalAsync Tareas.

Supongamos que usted tiene páginas denominadas Página principal y Infopage. Como el nombre sugiere, Página principal usos PushAsync para desplazarse hasta infopage para obtener alguna información del usuario, y de alguna manera infopage debe transferir esa información a Página principal.

Aquí hay algunas maneras que Página principal y infopage pueden interactuar (o no interactuar):

Página principal puede acceder a una propiedad en infopage o llamar a un método en el infopage después de crear instancias En-foPage o después de la PushAsync tarea se completa. Esto es sencillo, y ya se vio un ejemplo en el SinglePageNavigation programa.

infopage puede acceder a una propiedad en Página principal o llamar a un método en el Página principal en cualquier momento durante su existencia. De forma más conveniente, infopage puede realizar estas operaciones durante su OnAppearing anular (para inicialización) o la OnDisappearing anular (para preparar los valores finales). Durante la duración de su existencia, infopage puede obtener el Página principal ejemplo de la NavigationStack colección. Sin embargo, dependiendo del orden del OnAppearing y OnDisappearing llamadas en relación con la realización de la PushAsync o PopAsync Tareas, Página principal podría ser el último elemento de la NavigationStack,

o infopage podría ser el último elemento de la NavigationStack, en ese caso Página principal es el penúltimo elemento.

Página principal puede ser informado de que infopage ha devuelto el control al Página principal anulando su OnAppearing método. (Pero hay que tener en cuenta que este método no es llamado en los dispositivos Android cuando una

página modal ha vuelto de nuevo a la página que lo invocó.) Sin embargo, durante el OnAppearing anulación de Página principal, Página principal no puede ser completamente seguro de que la instancia de infopage se encuentra todavía en la NavigationStack colección, o incluso que existe en absoluto. Página principal puede guardar la instancia de info-Página cuando se navega a infopage, sino que crea problemas si la aplicación necesita para guardar el estado de la página cuando se termina.

Vamos a examinar un programa llamado DataTransfer1 que utiliza una segunda página para obtener información del usuario y luego añade esa información como un elemento a una Vista de la lista. El usuario puede añadir varios artículos a la Vista de la lista o editar un elemento existente con golpecitos. Para centrarse exclusivamente en el mecanismo de la comunicación Interpage, el programa no utiliza los enlaces de datos, y la clase que almacena la información no implementa INotifyPropertyChanged:

```
clase pública Información
{
    public string Nombre { conjunto ; obtener ; }

    public string Email { conjunto ; obtener ; }

    public string Idioma { conjunto ; obtener ; }

    público Fecha y hora Fecha { conjunto ; obtener ; }

    público cadena de anulación Encadenar()
    {
        regreso Cuerda .Formato( "{0} / {1} / {2} / {3: d}" ,
            Cuerda .IsNullOrEmptySpace (Nombre)? "???": Nombre,
            Cuerda .IsNullOrEmptySpace (correo electrónico)? "???": Email,
            Cuerda .IsNullOrEmptySpace (Idioma)? "???": Idioma,
            Fecha);
    }
}
```

los Encadenar método permite la Vista de la lista para mostrar los elementos con el mínimo esfuerzo.

los DataTransfer1 programa tiene dos páginas, llamado DataTransfer1HomePage y DataTransfer1InfoPage, que se comunican entre sí mediante llamadas a métodos públicos. los DataTransfer1HomePage tiene un archivo XAML con una Botón para invocar una página para obtener información y una Vista de la lista para la visualización de cada elemento y permitir que un elemento a ser editado:

```
< Página de contenido xmlns = "http://xamarin.com/schemas/2014/forms"
    xmlns : X = "http://schemas.microsoft.com/winfx/2009/xaml"
    X : Clase = "DataTransfer1.DataTransfer1HomePage"
    Título = "Página inicial">

    < Cuadrícula >
        < Botón Texto = "Agregar nuevo elemento"
            Grid.Row = "0"
            Tamaño de fuente = "Large"
            HorizontalOptions = "Centro"
            VerticalOptions = "Centro"
            hecho clic = "OnGetInfoButtonClicked" />
```

```

< Vista de la lista X : Nombre = "ListView"
  Grid.Row = "1"
  itemSelected = "OnListViewItemSelected" />

</ Cuadricula >
</ Pagina de contenido >

```

Vamos a rebotar hacia atrás y adelante entre las dos clases de examinar la transferencia de datos. Aquí está la parte del archivo de código subyacente que muestra la inicialización de la Vista de la lista con un ObservableCollection de manera que la Vista de la lista actualiza su presentación siempre que cambia la colección:

```

pública clase parcial DataTransfer1HomePage : Pagina de contenido
{
    ObservableCollection < Información > List = nuevo ObservableCollection < Información > ();

    pública DataTransfer1HomePage ()
    {
        InitializeComponent ();

        // Establecer la colección de ListView.
        listView.ItemsSource = lista;
    }

    // Botón Seguido manejador.
    vacío asíncrono OnGetInfoButtonClicked ( objeto remitente, EventArgs args )
    {
        esperar Navigation.PushAsync ( nuevo DataTransfer1InfoPage () );
    }
    ...
}

```

Este código también implementa la hecho clic controlador para el Botón simplemente creando una instancia de la DataTransfer1InfoPage y navegando a la misma.

El archivo XAML de DataTransfer1InfoPage tiene dos Entrada elementos, una Recogedor, y una Fecha-Recogedor correspondiente a las propiedades de Información. Esta página se basa en la interfaz de usuario estándar de cada plataforma para regresar a la página principal:

```

< Pagina de contenido xmlns = "http://xamarin.com/schemas/2014/forms"
  xmlns : X = "Http://schemas.microsoft.com/winfx/2009/xaml"
  X : Clase = "DataTransfer1.DataTransfer1InfoPage"
  Título = "Página de Información">

    < StackLayout Relleno = "20, 0"
      Espaciado = "20">
        < Entrada X : Nombre = "NameEntry"
          marcador de posición = "Entrar nombre" />

        < Entrada X : Nombre = "EmailEntry"
          marcador de posición = "Introducir dirección de correo electrónico" />

        < Recogedor X : Nombre = "LanguagePicker"
          Título = "Lenguaje de programación favorito">
            < Picker.Items >

```

```

<X : Cuerda > DO#</X : Cuerda >
<X : Cuerda > F#</X : Cuerda >
<X : Cuerda > C objetivo </X : Cuerda >
<X : Cuerda > Rápido </X : Cuerda >
<X : Cuerda > Java </X : Cuerda >

</ Picker.Items >
</ Recogedor >

< Selector de fechas X : Nombre = "Selector de fechas" />
</ StackLayout >
</ Página de contenido >

```

El archivo de código subyacente de la página de información de una instancia **Información objeto** que está asociado a esta instancia de página:

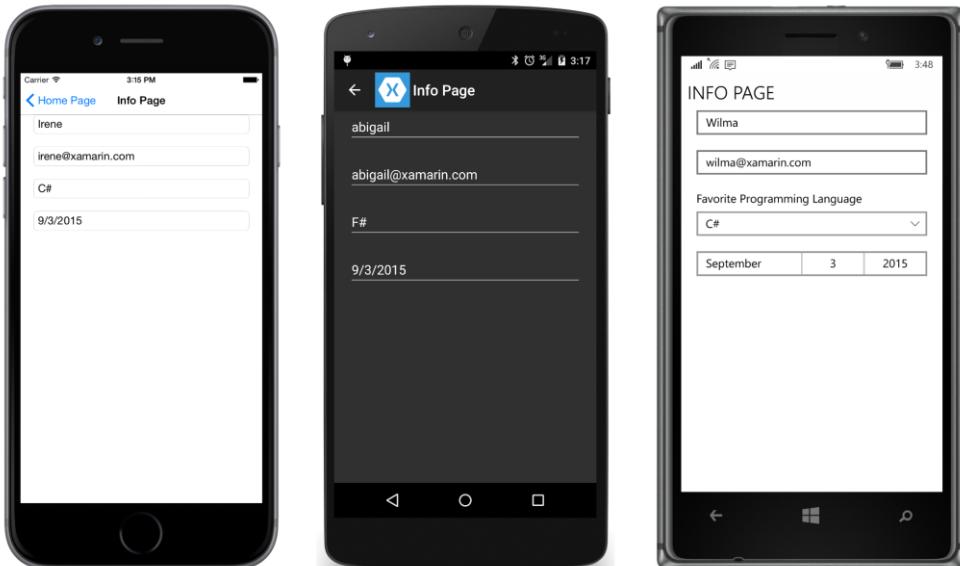
```

pública clase parcial DataTransfer1InfoPage : Página de contenido
{
    // instancia de un objeto de información para esta instancia de página.
    Información info = nuevo Información();

    pública DataTransfer1InfoPage ()
    {
        InitializeComponent ();
    }
    ...
}

```

El usuario interactúa con los elementos de la página introduciendo información:



Ninguna otra cosa sucede en la clase hasta **DataTransfer1InfoPage** recibe una llamada a su **OnDisappear**-**En g anular**. Esto por lo general indica que el usuario ha pulsado la **Espalda** botón que es o bien parte de la

barra de navegación (en iOS y Android) o por debajo de la pantalla (en Android y Windows Phone).

Sin embargo, es posible tener en cuenta que en una plataforma ya no apoyado por Xamarin.Forms (Silverlight Windows Phone), OnDisappearing se llama cuando el usuario invoca la Recogedor o Selector de fechas,

y es posible que se ponga nervioso acerca de ser llamado en otras circunstancias en las plataformas actuales. Esto implica que no debe hacerse nada en el OnDisappearing anular que no se puede deshacer cuando

OnDisappearing se llama como parte de la navegación normal volver a la página principal. Esta es la razón por Datos-Transfer1InfoPage crea una instancia de su Información objetar cuando la página se crea por primera vez y no durante el OnDisappearing anular.

los OnDisappearing override establece las propiedades de la Información objeto a partir de los cuatro puntos de vista y, a continuación obtiene la instancia de DataTransfer1HomePage la invocada desde el Navegación-

Apilar colección. A continuación, llama a un método denominado InformationReady en esa página de inicio:

```
pública clase parcial DataTransfer1InfoPage : Pagina de contenido
{
    ...
    protegido override void OnDisappearing ()
    {
        base.OnDisappearing ();

        propiedades del objeto Information // Set.
        info.Name = nameEntry.Text;
        info.Email = emailEntry.Text;

        En t index = languagePicker.SelectedIndex;
        info.Language = indice == -1? nulo : languagePicker.Items [indice];

        info.Date = datePicker.Date;

        // Obtener el DataTransfer1HomePage que invocó esta página.
        NavigationPage navPage = ( NavigationPage )Solicitud .Current.MainPage;
        IReadOnlyList < Página > NavStack = navPage.Navigation.NavigationStack;
        En t lastIndex = navStack.Count - 1;
        DataTransfer1HomePage HomePage = navStack [lastIndex] como DataTransfer1HomePage ;

        Si (Página de inicio == nulo )
        {
            HomePage = navStack [lastIndex - 1] como DataTransfer1HomePage ;
        }
        // Información de transferencia de objetos a DataTransfer1HomePage.
        HomePage.InformationReady (info);
    }
}
```

los InformationReady método en el DataTransfer1HomePage comprueba si el Información objeto ya está en el ObservableCollection establecido en el Vista de la lista, y si es así, lo reemplaza. De lo contrario, se añade el objeto a esa colección:

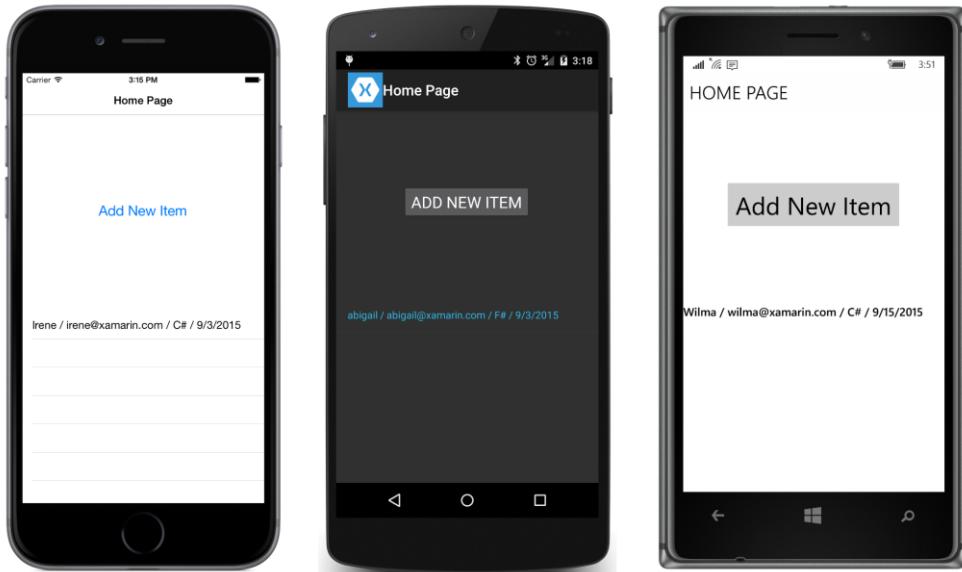
```
pública clase parcial DataTransfer1HomePage : Pagina de contenido
{
    ...
}
```

```
// Llamado de infopage.  
public void InformationReady ( Información info)  
{  
    // Si el objeto ya ha sido añadido, reemplazarlo.  
    En t index = list.IndexOf (info);  
  
    Si (índice! = -1)  
    {  
        lista [índice] = info;  
    }  
    // De lo contrario, añadirlo.  
    más  
    {  
        list.Add (info);  
    }  
}  
}
```

Hay dos motivos para comprobar si un Información objeto ya está en el Vista de la lista colección. Puede ser que sea ya, si la página de información recibido una llamada anterior a su OnDisappearing override, que luego se traduce en una llamada a InformationReady en la página principal. elementos también como podrá ver-existentes en el Vista de la lista se pueden editar.

El código que sustituye a la Información objeto con sí mismo en el ObservableCollection podría parecer superfluo. Sin embargo, el acto de sustituir el elemento hace que el ObservableCollection para disparar una CollectionChanged evento y el Vista de la lista vuelve a dibujar. Otra solución sería que En para-moción para implementar INotifyPropertyChanged, en cuyo caso el cambio en los valores de una propiedad haría que el Vista de la lista para actualizar la presentación de dicho elemento.

En este punto, estamos de vuelta en la página principal, y el Vista de la lista muestra el elemento recién añadido:



Ahora puede aprovechar el Botón de nuevo para crear un nuevo elemento, o puede tocar un elemento existente en el **Vista de la lista**. Los **itemSelected** controlador para el **Vista de la lista** También navega hasta **DataTransfer1Info-Página**:

```
público clase parcial DataTransfer1HomePage : Pagina de contenido
{
    ...
    // ListView itemSelected manejar.
    vecio asíncrono On.listViewItemSelected ( objeto remitente, SelectedItemChangedEventArgs args)
    {
        Si (Args.SelectedItem = nulo )
        {
            // Desactive el artículo.
            listView.SelectedItem = nulo ;

            DataTransfer1InfoPage infopage = nuevo DataTransfer1InfoPage ();
            esperar Navigation.PushAsync (infopage);
            infoPage.InitializeInfo (( Información ) Args.SelectedItem);
        }
    }
    ...
}
```

Sin embargo, después de la PushAsync tarea se completa, el controlador llama a un método en **DataTransfer1Info-Página** llamado **InitializeInfo** con el elemento seleccionado.

Los **InitializeInfo** método en el **DataTransfer1InfoPage** sustituye a la **Información** objeto que originalmente creado como un campo con esta instancia existente e inicializa las vistas en la página con las propiedades del objeto:

```
público clase parcial DataTransfer1InfoPage : Pagina de contenido
```

```
{  
...  
    public void InitializeInfo ( Información info)  
    {  
        // Reemplazar la instancia.  
        esta .info = información;  
  
        // Inicializar los puntos de vista.  
        nameEntry.Text = info.Name ?? "" ;  
        emailEntry.Text = info.Email ?? "" ;  
  
        Si (I Cuerda .IsNullOrEmpty (info.Language))  
        {  
            languagePicker.SelectedIndex = languagePicker.Items.IndexOf (info.Language);  
        }  
        datePicker.Date = info.Date;  
    }  
...  
}
```

Ahora el usuario está editando un elemento existente en vez de una nueva instancia.

En general, un programa que permite la edición de artículos existentes también se le dará al usuario la oportunidad de cancelar los cambios ya realizados en el elemento. Para permitir esto, el DataTransfer1InfoPage habría que diferenciar entre volver de nuevo a la página principal con los cambios y cancelar la operación de edición. Al menos uno Botón o ToolbarItem se requiere, y que probablemente debería haber una Cancelar

Botón de modo que la norma **Espalda** botón guarda los cambios.

Dicho programa también debe tener una instalación para eliminar elementos. Más adelante en este capítulo, verá un programa de este tipo.

El centro de mensajería

Usted no le gusta la idea de las dos clases de página método de realizar llamadas directamente entre sí. Parece que funciona bien para una muestra pequeña, pero para un programa más amplio con una gran cantidad de comunicación entre clases, es posible que prefiera algo un poco más flexible que no requiere de instancias de página reales.

una instalación de este tipo es el Xamarin.Forms MessagingCenter clase. Esta es una clase estática con tres métodos, llamado Suscribir, darse de baja, y Enviar. Los mensajes se identifican con una cadena de texto y pueden ir acompañados de cualquier objeto. Los Enviar método difunde un mensaje que es recibido por todos los abonados a ese mensaje.

los DataTransfer2 programa tiene la misma Información clase y los mismos archivos XAML como DataTransfer1, pero utiliza el MessagingCenter clase en lugar de llamadas a métodos directos.

El constructor de la página principal se suscribe a un mensaje identificado por la cadena de texto "InformationReady." Los argumentos genéricas a Suscribir indicar qué tipo de objeto envía este mensaje-un objeto de tipo DataTransfer2InfoPage -y el tipo de los datos, que es Información. los Suscribir argumentos del método indican el objeto de recibir el mensaje (esta), el nombre del mensaje,

y una función lambda. El cuerpo de esta función lambda es el mismo que el cuerpo de la En para-
mationReady método en el programa anterior:

```
público clase parcial DataTransfer2HomePage : Pagina de contenido
{
    ObservableCollection < Información > List = nuevo ObservableCollection < Información > ();

    público DataTransfer2HomePage ()
    {
        InitializeComponent ();

        // Establecer la colección de ListView.
        listView.ItemsSource = lista;

        // Suscribirse al mensaje "InformationReady".
        MessagingCenter.Subscribe < DataTransfer2InfoPage , Información >
            ( esta , "InformationReady" , ( Remitente, información )=>
        {
            // Si el objeto ya ha sido añadido, reemplazarlo.
            En t index = list.IndexOf ( info );

            Si ( Índice! = -1 )
            {
                lista [ index ] = info;
            }
            // De lo contrario, añadirlo.
            más
            {
                list.Add ( info );
            }
        });
    }

    // Botón Seguido manejador.
    vacío asíncrono OnGetInfoButtonClicked ( objeto remitente, EventArgs args)
    {
        esperar Navigation.PushAsync ( nuevo DataTransfer2InfoPage () );
    }

    // ListView itemSelected manejador.
    vacío asíncrono OnListViewItemSelected ( objeto remitente, SelectedItemChangedEventArgs args)
    {
        Si ( Args.SelectedItem! = nulo )
        {
            // Desactive el artículo.
            listView.SelectedItem = nulo ;

            DataTransfer2InfoPage infopage = nuevo DataTransfer2InfoPage ();
            esperar Navigation.PushAsync ( infopage );

            // Enviar mensaje "InitializeInfo" a las páginas de información.
            MessagingCenter.send < DataTransfer2HomePage , Información >
                ( esta , "InitializeInfo" , ( Información ) Args.SelectedItem );
        }
    }
}
```

```
    }  
}
```

los itemSelected manejador de la Vista de la lista contiene una llamada a MessagingCenter.Send. Los argumentos genéricos indican el tipo del remitente del mensaje y el tipo de los datos. Los argumentos del método indican el objeto de enviar el mensaje, el nombre del mensaje y los datos, que es el ítem seleccionado del Vista de la lista.

Los DataTransfer2InfoPage archivo de código subyacente contiene llamadas complementarias a Mensajería-Center.Subscribe y MessageCenter.Send. El constructor de información de la página se suscribe al mensaje "InitializeInfo"; el cuerpo de la función lambda es la misma que la InitializeInfo método en el programa anterior excepto que termina con un llamado a dejar de recibir el mensaje. Darse de baja asegura que ya no hay una referencia al objeto de información de la página y permite que el objeto info página sea basura recogida. En sentido estricto, sin embargo, darse de baja no debería ser necesario porque el men-

sagingCenter mantiene WeakReference objetos para los suscriptores:

```
publica clase parcial DataTransfer2InfoPage : Página de contenido  
{  
    // instancia de un objeto de información para esta instancia de página.  
    Información info = nuevo Información();  
  
    público DataTransfer2InfoPage ()  
    {  
        InitializeComponent ();  
  
        // Suscribirse al mensaje "InitializeInfo".  
        MessagingCenter.Subscribe < DataTransfer2HomePage , Información >  
            (esta , "InitializeInfo" , (Remitente, información) =>  
            {  
                // Reemplazar la instancia.  
                esta.info = información;  
  
                // Inicializar los puntos de vista.  
                nameEntry.Text = info.Name ?? "" ;  
                emailEntry.Text = info.Email ?? "" ;  
  
                Si (I Cuerda .IsNullOrEmpty (info.Language))  
                {  
                    languagePicker.SelectedIndex = languagePicker.Items.IndexOf (info.Language);  
                }  
                datePicker.Date = info.Date;  
  
                // No necesite "InitializeInfo" más así que darse de baja.  
                MessagingCenter.Unsubscribe < DataTransfer2HomePage , Información >  
                    (esta , "InitializeInfo" );  
            });  
    }  
  
    protegido override void OnDisappearing ()  
    {  
        base.OnDisappearing ();
```

```

propiedades del objeto Information // Set.
info.Name = nameEntry.Text;
info.Email = emailEntry.Text;

En t index = languagePicker.SelectedIndex;
info.Language = índice == -1? nulo : languagePicker.Items [índice];

info.Date = datePicker.Date;

// Enviar mensaje "InformationReady" volver a la página principal.
MessagingCenter .send < DataTransfer2InfoPage , Información >
( esta , "InformationReady" , Información);
}
}

```

los OnDisappearing la redefinición es considerablemente más corta que la versión en el programa anterior. Para llamar a un método en la página de inicio, el programa anterior tuvo que entrar en el NavigationStack colección. En esta versión, todo lo que es necesario es utilizar MessagingCenter.Send para enviar un mensaje de "InformationReady" a todo el que se ha suscrito a él, y que pasa a ser la página principal.

Eventos

En tanto el enfoque método de llamada y el enfoque de mensajería en el centro de la comunicación entre clases, la página de información necesita conocer el tipo de la página principal. Esto a veces es indeseable si la misma páginas de información se puede llamar de diferentes tipos de páginas.

Una solución a este problema es para la clase de información para implementar un evento, y ese es el enfoque adoptado en **DataTransfer3**. Los **Información archivos de clase y XAML** son los mismos que los programas anteriores, pero **DataTransfer3InfoPage** ahora implementa un evento público denominado **InformationReady**:

```

público clase parcial DataTransfer3InfoPage : Pagina de contenido
{
    // Definir un evento público para la transferencia de datos.
    público Controlador de eventos < Información > InformationReady;

    // instancia de un objeto de información para esta instancia de página.
    Información info = nuevo Información ();

    público DataTransfer3InfoPage ()
    {
        InitializeComponent ();
    }

    public void InitializeInfo ( Información info )
    {
        // Reemplazar la instancia.
        esta .info = información;

        // Inicializar los puntos de vista.
        nameEntry.Text = info.Name ?? "";
        emailEntry.Text = info.Email ?? "";
    }
}

```

```

Si (I Cuerda .IsNullOrEmptyWhiteSpace (info.Language))
{
    languagePicker.SelectedIndex = languagePicker.Items.IndexOf (info.Language);
}
datePicker.Date = info.Date;
}

protegido override void OnDisappearing ()
{
    base .OnDisappearing ();

    propiedades del objeto Information // Set.
    info.Name = nameEntry.Text;
    info.Email = emailEntry.Text;

    En t index = languagePicker.SelectedIndex;
    info.Language = índice == -1? nulo : languagePicker.Items [índice];

    info.Date = datePicker.Date;

    // provocar el evento InformationReady.
    Controlador de eventos < Información > Handler = InformationReady;

    Si (Manejador! = nulo )
        entrenador de animales( esta , Información);
    }
}
}

```

Durante el OnDisappearing anula, la clase establece el Información propiedades de los elementos y plantea una InformationReady evento con el Información objeto.

La página principal se puede configurar un controlador para el InformationReady evento o bien después de que se crea una instancia de la página de información o después de que se desplaza a la página. El controlador de eventos añade el Información oponerse a la

Vista de la lista o reemplaza un elemento existente:

```

pública clase parcial DataTransfer3HomePage : Pagina de contenido
{
    ObservableCollection < Información > List = nuevo ObservableCollection < Información > ();

    pública DataTransfer3HomePage ()
    {
        InitializeComponent ();

        // Establecer la colección de ListView.
        listView.ItemsSource = lista;
    }

    // Botón Seguido manejador.
    vacío asíncrono OnGetInfoButtonClicked ( objeto remitente, EventArgs args)
    {
        DataTransfer3InfoPage infopage = nuevo DataTransfer3InfoPage ();
        esperar Navigation.PushAsync (infopage);

        // Establecer controlador de eventos para obtener información.
    }
}

```

```
infoPage.InformationReady += OnInfoPageInformationReady;
}

// ListView itemSelected manejador.
vacío async OnListViewItemSelected ( objeto remitente, SelectedItemChangedEventArgs args)
{
    Si (Args.SelectedItem == nulo )
    {
        // Desactive el artículo.
        listView.SelectedItem = nulo ;

        DataTransfer3InfoPage infopage = nuevo DataTransfer3InfoPage ();
        esperar Navigation.PushAsync (infopage);
        infoPage.InitializeInfo (( Información ) Args.SelectedItem);

        // Establecer controlador de eventos para obtener información.
        infoPage.InformationReady += OnInfoPageInformationReady;
    }
}

vacío OnInfoPageInformationReady ( objeto remitente, Información info)
{
    // Si el objeto ya ha sido añadido, reemplazarlo.
    En t index = list.IndexOf (info);

    Si (indice! = -1)
    {
        lista [index] = info;
    }
    // De lo contrario, añadirlo.
    más
    {
        list.Add (info);
    }
}
}
```

Hay un par de problemas con este enfoque. El primer problema es que no hay un lugar conveniente para separar el controlador de eventos.

La página de información genera el evento en su OnDisappearing anular. Si no está seguro de que OnDisappearing se llama sólo cuando se está produciendo la navegación, a continuación, la página principal no se puede separar el lanzador evento en el propio controlador.

Tampoco puede la página principal de separar el controlador de eventos en su OnAppearing anulación porque cuando la página de información vuelve de nuevo a la página principal, el orden en el cual el OnAppearing y OnDisappearing anulaciones se llaman depende de la plataforma.

Si la página de inicio no puede separar el manejador de la página de información, entonces cada instancia de las páginas de información seguirá manteniendo una referencia a la página principal y no puede ser basura recogida.

También el enfoque de controlador de eventos no es bueno cuando una aplicación necesita para guardar y restaurar el estado de la página. La página de información no puede salvar el estado del evento para restaurarlo cuando el programa se ejecuta de nuevo.

El intermediario clase de aplicaciones

En una aplicación Xamarin.Forms, el primer código que se ejecuta en el proyecto de código común es el constructor de una clase denominada habitualmente **Aplicación que deriva de Solicitud**. Esta Aplicación objeto permanece constante hasta que el programa termina, y siempre está disponible para cualquier código en el programa a través de la estática Application.Current propiedad. El valor de retorno de esa propiedad es de tipo aplicabilidad cién, pero es fácil de echarlo a App.

Esto implica que el Aplicación clase es un gran lugar para almacenar los datos que deben ser accesibles en toda la aplicación, incluidos los datos que se transfiere de una página a otra.

los Información clase en el DataTransfer4 versión del programa es un poco diferente de la versión que usted ha visto anteriormente:

```
clase pública Información
{
    público Información()
    {
        fecha = Fecha y hora .Hoy;
    }

    public string Nombre { conjunto ; obtener ; }

    public string Email { conjunto ; obtener ; }

    public string Idioma { conjunto ; obtener ; }

    público Fecha y hora Fecha { conjunto ; obtener ; }

    público cadena de anulación Encadenar()
    {
        regreso Cuerda .Formato( "{0} / {1} / {2} / {3: d}" ,
            Cuerda .IsNullOrEmptySpace (Nombre)? "???" : Nombre,
            Cuerda .IsNullOrEmptySpace (correo electrónico)? "???" : Email,
            Cuerda .IsNullOrEmptySpace (Idioma)? "???" : Idioma,
            Fecha);
    }
}
```

El constructor de esta versión establece el Fecha propiedad a la fecha de hoy. En las versiones anteriores del programa, las propiedades de una Información instancia se establecen a partir de los diversos elementos de la página de información. En ese caso, el Fecha propiedad se establece a partir de la Selector de fechas, el cual por conjuntos predeterminados de su Fecha propiedad a la fecha actual. En DataTransfer4, como se verá, los elementos en la página de información se inicializan de las propiedades en el Información objeto, por lo que establecer la Fecha propiedad en el Información clase simplemente mantiene la funcionalidad de los programas consistentes.

Aquí está la Aplicación clase de DataTransfer4. Fíjese en las propiedades públicas nombradas InfoCollection y CurrentInfoItem. Que se inicie la constructor InfoCollection a una ObservableCollection <información> objeto antes de crear DataTransfer4HomePage:

```
clase pública Aplicación : Solicitud
```

```
{  
    público App ()  
    {  
        // Crear el ObservableCollection de los elementos de información.  
        InfoCollection = nuevo ObservableCollection < Información > ();  
  
        MainPage = nuevo NavigationPage ( nuevo DataTransfer4HomePage () );  
    }  
  
    público IList < Información > InfoCollection { conjunto privado ; obtener ; }  
  
    público Información CurrentInfoItem { conjunto ; obtener ; }  
    ...  
}
```

La disponibilidad de la InfoCollection bienes en Aplicación permite DataTransfer4HomePage para fijar directamente a la ItemsSource propiedad de la Vista de la lista:

```
público clase parcial DataTransfer4HomePage : Pagina de contenido  
{  
    Aplicación aplicación = ( Aplicación ) Solicitud .Corriente;  
  
    público DataTransfer4HomePage ()  
    {  
        InitializeComponent ();  
  
        // Establecer la colección de ListView.  
        listView.ItemsSource = app.InfoCollection;  
    }  
  
    // Botón Seguido manejador.  
    vacío asíncrono OnGetInfoButtonClicked ( objeto remitente, EventArgs args)  
    {  
        // Crear nuevo elemento de información.  
        app.CurrentInfoItem = nuevo Información ();  
  
        // Vaya a las páginas de información.  
        esperar Navigation.PushAsync ( nuevo DataTransfer4InfoPage () );  
    }  
  
    // ListView itemSelected manejador.  
    vacío asíncrono OnListViewItemSelected ( objeto remitente, SelectedItemChangedEventArgs args)  
    {  
        Si (Args.SelectedItem != nulo )  
        {  
            // Desactive el artículo.  
            listView.SelectedItem = nulo ;  
  
            // Obtener elemento de información existentes.  
            app.CurrentInfoItem = ( Información ) Args.SelectedItem;  
  
            // Vaya a las páginas de información.  
            esperar Navigation.PushAsync ( nuevo DataTransfer4InfoPage () );  
        }  
    }  
}
```

```
    }  
}
```

Fíjese en las dos maneras diferentes, pero similares que la hecho clic controlador para el Botón y el Ítem seleccionado controlador para el Vista de la lista son implementados. Antes de la navegación hacia DataTransfer4InfoPage, ambos manipuladores establecen el CurrentInfoItem propiedad de la Aplicación clase a una instancia de En para-moción. Pero el hecho clic establece el manejador CurrentInfoItem propiedad a una nueva instancia, mientras que el itemSelected manejador lo establece en el elemento seleccionado en el Vista de la lista.

Todo lo demás es manejado por DataTransfer4InfoPage. La página de información puede inicializar los elementos en la página de la información objeto almacenado en el CurrentInfoItem propiedad de la Aplicación clase:

```
público clase parcial DataTransfer4InfoPage : Página de contenido  
{  
    Aplicación aplicación = (Aplicación)Solicitud.Comiente;  
  
    público DataTransfer4InfoPage ()  
    {  
        InitializeComponent ();  
  
        // Inicializar los puntos de vista.  
        Información info = app.CurrentInfoItem;  
  
        nameEntry.Text = info.Name ?? "";  
        emailEntry.Text = info.Email ?? "";  
  
        Si (l Cuerda.IsNullOrWhiteSpace (info.Language))  
        {  
            languagePicker.SelectedIndex = languagePicker.Items.IndexOf (info.Language);  
        }  
        datePicker.Date = info.Date;  
    }  
  
    protegido override void OnDisappearing ()  
    {  
        base.OnDisappearing ();  
  
        propiedades del objeto Information // Set.  
        Información info = app.CurrentInfoItem;  
  
        info.Name = nameEntry.Text;  
        info.Email = emailEntry.Text;  
  
        En l index = languagePicker.SelectedIndex;  
        info.Language = índice == -1? nulo : languagePicker.Items [índice];  
  
        info.Date = datePicker.Date;  
  
        // Si el objeto ya ha sido añadido a la colección, reemplazarlo.  
        Il.list < Información > List = app.InfoCollection;  
  
        index = list.IndexOf (info);
```

```

    Si (indice! = -1)
    {
        lista [index] = info;
    }
    // De lo contrario, añadirlo.
    más
    {
        list.Add (info);
    }
}
}

```

La página de información todavía necesita para ignorar sus `OnDisappearing` método para establecer las propiedades de la En-
formación objetar y, posiblemente, añadirlo a la Vista de la lista colección o sustituir el mismo objeto para activar un redibujado. Pero la página de
información no tiene que acceder directamente a la Vista de la lista ya que puede obtener el
`ObservableCollection` desde el `InfoCollection` propiedad de la Aplicación clase.

Por otra parte, si usted necesita para guardar y restaurar el estado de la página, todo está disponible justo en el Aplicación clase.

Vamos a ver cómo eso podría funcionar.

El cambio a un modelo de vista

En este punto debería ser obvio que la Información clase realmente debe aplicar `INotifyPropertyChanged`. En `DataTransfer5`, el Información clase ha convertido en una `InformationViewModel` clase. Se deriva de `ViewModelBase` en el `Xamarin.FormsBook.Toolkit` biblioteca para reducir la sobrecarga:

```

público clase InformationViewModel : ViewModelBase
{
    cuerda nombre, correo electrónico, el idioma;
    Fecha y hora fecha = Fecha y hora .Hoy;

    público cuerda Nombre
    {
        conjunto SetProperty ( árbito nombre, valor );
        obtener { regreso nombre; }
    }

    público cuerda Email
    {
        conjunto SetProperty ( árbito correo electrónico, valor );
        obtener { regreso correo electrónico; }
    }

    público cuerda Idioma
    {
        conjunto SetProperty ( árbito idioma, valor );
        obtener { regreso idioma; }
    }

    público Fecha y hora Fecha

```

```

    {
        conjunto { SetProperty ( árbitro fecha, valor ); }
        obtener ( regreso fecha; )
    }
}
}

```

Una nueva clase se ha añadido a **DataTransfer5** llamado Datos de aplicación. Esta clase incluye una ObservableCollection de Información objetos para el Vista de la lista así como una separada Información instancia para la página de información:

```

clase pública Datos de aplicación
{
    público Datos de aplicación()
    {
        InfoCollection = nuevo ObservableCollection < InformationViewModel > ();
    }

    público IList < InformationViewModel > InfoCollection { conjunto privado ; obtener ; }

    público InformationViewModel Información actual { conjunto ; obtener ; }
}

```

los Aplicación crea una instancia de la clase Datos de aplicación antes de crear instancias de la página principal y la hace disponible como una propiedad pública:

```

clase pública Aplicación : Solicitud
{
    público App ()
    {
        // Asegúrese de enlace a Toolkit biblioteca.
        nuevo Xamarin.FormsBook.Toolkit.ObjectToIndexConverter < objeto > ();

        // Instantie datos de programa y establecer la propiedad.
        AppData = nuevo Datos de aplicación ();

        // Ir a la pagina principal.
        MainPage = nuevo NavigationPage ( nuevo DataTransfer5HomePage () );
    }

    público Datos de aplicación Datos de aplicación { conjunto privado ; obtener ; }
    ...
}

```

El archivo XAML de **DataTransfer5HomePage** establece el BindingContext de la página con una unión que incorpora la estática Application.Cu propiedad (que devuelve el Aplicación objeto) y la Datos de aplicación ejemplo. Esto significa que el Vista de la lista puede unirse a su ItemsSource alojamiento hasta el info-Colección propiedad de Datos de aplicación:

```

< Página de contenido xmlns = "http://xamarin.com/schemas/2014/forms"
    xmlns : X = "Http://schemas.microsoft.com/winfx/2009/xaml"
    X : Clase = "DataTransfer5.DataTransfer5HomePage"
    Título = "Página inicial"
    BindingContext = "{ Unión Fuente = { X : Estático Solicitud .Comiente },

```

```

Camino = AppData} >

< Cuadricula >
  < Botón Texto = "Agregar nuevo elemento" 
    Grid.Row = "0"
    Tamaño de fuente = "Large"
    HorizontalOptions = "Centro"
    VerticalOptions = "Centro"
    hecho clic = "OnGetInfoButtonClicked" />

  < Vista de la lista X : Nombre = "ListView" 
    Grid.Row = "1"
    ItemsSource = "{ Unión InfoCollection }"
    itemSelected = "OnListViewItemSelected">
    < ListView.ItemTemplate >
      < DataTemplate >
        < ViewCell >
          < StackLayout Orientación = "Horizontal" >
            < Etiqueta Texto = "{ Unión Nombre }"/>
            < Etiqueta Texto = "/" />
            < Etiqueta Texto = "{ Unión Email }"/>
            < Etiqueta Texto = "/" />
            < Etiqueta Texto = "{ Unión Idioma }"/>
            < Etiqueta Texto = "/" />
            < Etiqueta Texto = "{ Unión Fecha , StringFormat = '{0: d}' }"/>
          </ StackLayout >
        </ ViewCell >
      </ DataTemplate >
    </ ListView.ItemTemplate >
  </ Vista de la lista >
</ Cuadricula >
</ Pagina de contenido >

```

Las versiones anteriores del programa de confiar en el Encadenar SOBREPASO Información para mostrar los elementos. Ahora eso Información ha sido sustituido por InformationViewModel, el Encadenar método no es adecuado porque no hay notificación de que el Encadenar método podría devolver algo diferente. En cambio, el Vista de la lista utiliza una ViewCell que contiene elementos con enlaces a las propiedades de InformationViewModel.

El archivo de código subyacente sigue llevando a cabo la hecho clic controlador para el Botón y el ítem Seleccionado controlador para el Vista de la lista, pero ahora son tan similares que pueden hacer uso de un método común llamado GoToInfoPage:

```

público clase parcial DataTransfer5HomePage : Pagina de contenido
{
  público DataTransfer5HomePage ()
  {
    InitializeComponent ();
  }

  // Botón Seguido manejador.
  void OnGetInfoButtonClicked ( objeto remitente, EventArgs args )
  {
    // Vaya a la página de información.
  }
}

```

```
        GoToInfoPage ( nuevo InformationViewModel () , cierto ) ;
    }

    // ListView itemSelected manejador .
    vacío OnListViewItemSelected ( objeto remitente , SelectedItemChangedEventArgs args )
    {
        Si ( Args.SelectedItem = nulo )
        {
            // Desactive el artículo .
            listView.SelectedItem = nulo ;

            // Vaya a la página de información .
            GoToInfoPage ( ( InformationViewModel ) Args.SelectedItem , falso ) ;
        }
    }

    vacío asíncrono GoToInfoPage ( InformationViewModel información , bool isNewItem )
    {
        // Obtener datos de programa objeto ( establecido en BindingContext en el archivo XAML ) .
        Datos de aplicación appData = ( Datos de aplicación ) BindingContext ;

        // elemento Conjunto de información CurrentInfo propiedad de datos de programa .
        appData.CurrentInfo = información ;

        // Vaya a la página de información .
        esperar Navigation.PushAsync ( nuevo DataTransfer5InfoPage () );

        // Añadir nuevo elemento de información a la colección .
        Si ( isNewItem )
        {
            appData.InfoCollection.Add ( info );
        }
    }
}
```

Para ambos casos, el GoToInfoPage método establece el Información actual propiedad de Datos de aplicación. Para hecho clic evento, que está establecido en un nuevo InformationViewModel objeto. Para el itemSelected evento, que está establecido en una ya existente InformationViewModel desde el Vista de la lista colección. Los isNewItem parámetro de la GoToInfoPage método indica si esta InformationViewModel objeto también debe añadirse a la InfoCollection de Datos de aplicación.

Observe que el nuevo elemento se agrega a la InfoCollection después de la PushAsync tarea se completa. Si el elemento se añade antes de la PushAsync llamar, entonces dependiendo de la plataforma se puede notar este nuevo elemento aparece de repente en el Vista de la lista inmediatamente antes de la transición de página. Eso podría ser un poco molesto!

El archivo XAML para el DataTransfer5InfoPage establece el BindingContext de la página a la Información actual propiedad de Datos de aplicación. (La página de inicio establece el Información actual propiedad de Datos de aplicación antes de crear instancias de la página de información, así que no es necesario para Datos de aplicación para implementar INotifyPropertyChanged). El ajuste de la BindingContext permite que todos los elementos visuales de la página para estar unidos a las propiedades en el InformationViewModel clase:

```

< Pagina de contenido xmlns = "http://xamarin.com/schemas/2014/forms"
    xmlns : X = "Http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns : kit de herramientas =
        "Clr-espacio de nombres: Xamarin.FormsBook.Toolkit; montaje = Xamarin.FormsBook.Toolkit"
    X : Clase = "DataTransfer5.DataTransfer5InfoPage"
    Título = "Página de Información"
    BindingContext = "{ Unión Fuente = { X : Estático Solicitud .Corriente},
        Camino = AppData.CurrentInfo } ">

    < StackLayout Relleno = "20, 0"
        Espaciado = "20">
        < Entrada Texto = "{ Unión Nombre }"
            marcador de posición = "Entrar nombre" />

        < Entrada Texto = "{ Unión Email }"
            marcador de posición = "Introducir dirección de correo electrónico" />

        < Recogedor X : Nombre = "LanguagePicker"
            Título = "Lenguaje de programación favorito">
            < Picker.Items >
                < X : Cuerda > DO# </X : Cuerda >
                < X : Cuerda > F# </X : Cuerda >
                < X : Cuerda > C objetivo </X : Cuerda >
                < X : Cuerda > Rápido </X : Cuerda >
                < X : Cuerda > Java </X : Cuerda >
            </ Picker.Items >

            < Picker.SelectedIndex >
                < Unión Camino = "Idioma">
                    < Binding Converter >
                        < kit de herramientas : ObjectToIndexConverter X : TypeArguments = "X: String">
                            < X : Cuerda > DO# </X : Cuerda >
                            < X : Cuerda > F# </X : Cuerda >
                            < X : Cuerda > C objetivo </X : Cuerda >
                            < X : Cuerda > Rápido </X : Cuerda >
                            < X : Cuerda > Java </X : Cuerda >
                        </ kit de herramientas : ObjectToIndexConverter >
                    </ Binding Converter >
                </ Unión >
            </ Picker.SelectedIndex >
        </ Recogedor >

        < Selector de fechas Fecha = "{ Unión Fecha }"/>
    </ StackLayout >
</ Pagina de contenido >

```

Observe el uso de la `ObjectToIndexConverter` en la unión entre la `SelectedIndex` propiedad de la `Recogedor` y la cadena `Idioma` propiedad de `InformationViewModel`. Este convertidor de unión se introdujo en el capítulo 19, "vistas Collection", en la sección "Enlace de datos del Selector."

El archivo de código subyacente de `DataTransfer5InfoPage` logra el objetivo MVVM de ser nada más que una llamada a `InitializeComponent`:

```
público clase parcial DataTransfer5InfoPage : Pagina de contenido
```

```
{  
    público DataTransfer5InfoPage ()  
    {  
        InitializeComponent ();  
    }  
}
```

El otro aspecto conveniente de **DataTransfer5** es que ya no hay una necesidad de sobreescribir **EnterPage**, **LeavePage**, **Appearing** y **OnDisappearing** métodos, y no hay necesidad de preguntarse por el orden de estas llamadas a métodos durante la navegación de páginas.

Pero lo que es realmente agradable es que es fácil de migrar **DataTransfer5** a una versión que guarda los datos de aplicación cuando se termina el programa y lo restaura la próxima vez que se ejecute el programa.

Guardado y restablecimiento de estado de la página

En particular, a medida que comience a trabajar más con las aplicaciones de varias páginas, es muy beneficioso para el tratamiento de las páginas de su aplicación *no* como los principales repositorios de datos, pero vistas visuales e interactivos meramente como temporales de los datos subyacentes. La palabra clave aquí es *temporal*. Si se mantiene los datos subyacentes hasta la fecha como el usuario interactúa con él, a continuación, las páginas pueden aparecer y desaparecer sin preocuparse.

El programa final de esta serie es **DataTransfer6**, lo que ahorra los contenidos de Datos de aplicación (y alguna otra información) en la solicitud de almacenamiento local cuando el programa se suspende y por lo tanto, cuando se termina y el programa recupera entonces que los datos de la próxima vez que el programa se inicia.

Además de guardar los datos que el usuario ha introducido con esmero, es probable que también deseá guardar el estado de la pila de navegación de páginas. Esto significa que si el usuario está introduciendo datos en la página de información y el programa termina, entonces la próxima vez que el programa se ejecuta, que navega a la página de información con los datos introducidos parcialmente restaurado.

Como se recordará, el **Solicitud** clase define una propiedad denominada **propiedades** que es un diccionario con cuerdas llaves y objeto valores. Puede configurar los elementos de la propiedades diccionario de antes o durante el **OnSleep** anular en su **Aplicación** clase. Los artículos estarán disponibles la próxima vez que el **Aplicación** se ejecuta el constructor.

La plataforma subyacente serializa objetos en el **propiedades** diccionario mediante la conversión de los objetos a un formulario en el que se pueden guardar en un archivo. No importa que el programador de la aplicación si se trata de una forma binaria o una forma de cadena, tal vez XML o JSON.

Para los números enteros o de punto flotante, para Fecha y hora valores, o por cadenas, la serialización es sencillo. En algunas plataformas, que podría ser posible para salvar a una instancia de una clase más compleja, como **InformationViewModel**, directamente a la **propiedades colección**. Sin embargo, esto no funciona en todas las plataformas. Es mucho más seguro para serializar clases de sí mismo para cadenas XML o JSON y luego guardar las cadenas resultantes de la **propiedades colección**. Con la versión de .NET disponible para **Xamarin.Forms** bibliotecas de clases portátiles, serialización XML es un poco más fácil que la serialización JSON, y eso es lo **DataTransfer6** usos.

Al realizar la serialización y deserialización, es necesario tener cuidado con referencias a objetos. Serialización no conserva igualdad de objetos. Vamos a ver cómo esto puede ser un problema:

La versión de Datos de aplicación introducido en DataTransfer5 tiene dos propiedades: InfoCollection, que es una colección de InformationViewModel objetos, y Información actual, que es una Información-ViewModel objeto que se está editando en ese momento.

El programa se basa en el hecho de que la Información actual objeto es también un elemento de la InfoCollection. los Información actual se convierte en el BindingContext para la página de información, y las propiedades de ese InformationViewModel instancia se interactivamente alterada por el usuario. Pero sólo porque ese mismo objeto es parte de InfoCollection serán los nuevos valores aparecen en el Vista de la lista.

¿Qué pasa cuando serializa el InfoCollection y Información actual propiedades de Datos de aplicación y luego deserializar para crear un nuevo ¿Datos de aplicación?

En la versión deserializado, la Información actual objeto tendrá las mismas propiedades exactas como uno de los elementos de la InfoCollection, pero no va a ser la misma instancia. Si el programa se restaura para permitir al usuario continuar la edición de un artículo en la página de información, ninguna de esas modificaciones se verá reflejada en el objeto en el Vista de la lista colección.

Con esa preparación mental, ahora es el momento para mirar la versión de Datos de aplicación en DataTransfer6.

```
público clase Datos de aplicación
{
    público Datos de aplicación()
    {
        InfoCollection = nuevo ObservableCollection < InformationViewModel >();
        CurrentInfoIndex = -1;
    }

    público ObservableCollection < InformationViewModel > InfoCollection { conjunto privado ; obtener ; }

    [ XmlIgnore ]
    público InformationViewModel Información actual { conjunto ; obtener ; }

    public int CurrentInfoIndex { conjunto ; obtener ; }

    público cuerda Publicar por fascículos()
    {
        // Si el CurrentInfo es válido, establecer el CurrentInfoIndex.
        Si (CurrentInfo == nulo )
        {
            CurrentInfoIndex = InfoCollection.IndexOf (CurrentInfo);
        }
        XmlSerializer serializador = nuevo XmlSerializer ( tipo de ( Datos de aplicación ));
        utilizando ( StringWriter StringWriter = nuevo StringWriter ())
        {
            serializer.Serialize (StringWriter, esta );
            regreso StringWriter.GetStringBuilder () ToString () ;
        }
    }
}
```

```

public static Datos de aplicación deserializar ( cuerda strAppData)
{
    XmlSerializer serializador = nuevo XmlSerializer ( tipo de ( Datos de aplicación ));
    utilizando ( StringReader StringReader = nuevo StringReader (StrAppData))
    {
        Datos de aplicación appData = ( Datos de aplicación ) Serializer.Deserialize (StringReader);

        // Si el CurrentInfoIndex es válido, establecer el CurrentInfo.
        Si (AppData.CurrentInfoIndex = -1)
        {
            appData.CurrentInfo = appData.InfoCollection [appData.CurrentInfoIndex];
        }
        regreso datos de aplicación;
    }
}
}

```

Esta versión cuenta con una InfoCollection propiedad y una Información actual propiedad como la versión anterior, pero también incluye una CurrentInfoIndex propiedad de tipo Entero, y el Información actual la propiedad se encuentra en posición con el XmlIgnore atributo, lo que significa que no va a ser serializada.

La clase también tiene dos métodos, denominados Publicar por fascículos y Deserializar. Publicar por fascículos comienza estableciendo la CurrentInfoIndex propiedad para el índice de Información actual dentro de InfoCollection. A continuación, convierte la instancia de la clase a una cadena XML y devuelve esa cadena.

Deserialize hace lo contrario. Es un método estático con una cuerda argumento. La cadena se supone que es la representación XML de una Datos de aplicación objeto. Después de que se convierte en una Datos de aplicación ejemplo, el método establece el Información actual propiedad basada en el CurrentInfoIndex propiedad. Ahora

Información actual es una vez más el objeto idéntico a uno de los miembros de la InfoCollection. El método devuelve que Datos de aplicación ejemplo.

El único otro cambio de DataTransfer5 a DataTransfer6 es el Aplicación clase. Los OnSleep override serializa el Datos de aplicación objeto y lo guarda en el propiedades diccionario con una tecla de "appData". Sino que también ahorra un valor booleano con la tecla "isInfoPageActive" si el usuario ha navegado a Datos-Transfer6InfoPage y es, posiblemente, en el proceso de introducir o editar información.

Los Aplicación constructor de la cadena deserializa disponible en la "appData" propiedades entrada o establece el Datos de aplicación propiedad a una nueva instancia si no existe esa entrada de diccionario. Si la entrada "isInfoPageActive" es cierto, no sólo debe crear una instancia DataTransfer6MainPage como el argumento de la navegaciónPage constructor (como es habitual), sino que también debe navegar hasta DataTransfer6InfoPage:

```

clase pública Aplicación : Solicitud
{
    público App ()
    {
        // Asegúrese de enlace a Toolkit biblioteca.
        Xamarin.FormsBook.Toolkit.Toolkit .En eso;

        // Cargar datos de programa anterior, si existe.
    }
}

```

```
Si (Properties.ContainsKey ( "datos de aplicación" ))
{
    AppData = Datos de aplicación .Deserialize ( ( cuerda ) Propiedades [ "datos de aplicación" ]);
}
más
{
    AppData = nuevo Datos de aplicación ();
}

// Lanzamiento de la página de inicio.
Página homepage = nuevo DataTransfer6HomePage ();
MainPage = nuevo NavigationPage (página principal);

// Es posible que vaya a las páginas de información.
Si (Properties.ContainsKey ( "IsInfoPageActive" ) &&
    ( bool ) Propiedades [ "IsInfoPageActive" ])
{
    homePage.Navigation.PushAsync ( nuevo DataTransfer6InfoPage (0, falso );
}
}

público Datos de aplicación Datos de aplicación { conjunto privado ; obtener ; }

protegido override void OnStart ()
{
    // manipulador cuando se inicia el app
}

protegido override void OnSleep ()
{
    // Guardar datos de programa serializa en cadena.
    propiedades [ "datos de aplicación" ] = AppData.Serialize ();

    // Guardar booleana para las páginas de información activa.
    propiedades [ "IsInfoPageActive" ] =
        MainPage.Navigation.NavigationStack.Last () es DataTransfer6InfoPage ;
}

protegido override void En resumen()
{
    // manejar cuando sus hojas de vida de aplicaciones
}
```

Para probar este programa, es necesario para terminar el programa de tal manera que la Aplicación clase recibe una llamada a su OnSleep método. Si está ejecutando el programa bajo el Visual Studio o Xamarin Studio depurador, hacer *no* terminar el programa desde el depurador. En su lugar, terminar la aplicación en el teléfono.

Quizás la mejor manera de poner fin a un programa en los teléfonos y emuladores de teléfono es mostrar primero todos los programas actualmente en ejecución:

- En iOS, toca dos veces la Casa botón.

- En Android, toque el (más a la derecha) **Tarea múltiple** botón.
- En Windows Phone, mantenga presionado el (más a la izquierda) **Espalda** botón.

Esta acción hace que el `OnSleep` método que se llama. A continuación, puede terminar el programa:

- En iOS, la aplicación deslizar hacia arriba.
- En Android, deslizar hacia un lado.
- En Windows Phone, deslizar hacia abajo.

Cuando se ejecuta el programa de Windows en una ventana, puede terminar el programa, simplemente haciendo clic en el **Cerca** botón. En el modo comprimido, desliza el programa desde la parte superior.

A continuación, puede utilizar Visual Studio o Xamarin Estudio para detener la depuración de la aplicación (si es necesario). A continuación, ejecute el programa de nuevo para ver si se "recuerda" donde lo dejó.

Guardado y restablecimiento de la pila de navegación

Muchas aplicaciones de varias páginas tienen una arquitectura de página que es más complejo de lo **DataTransfer6**, y usted desea una forma generalizada para guardar y restaurar toda la pila de navegación. Por otra parte, es probable que desee integrar la conservación de la pila de navegación con una forma sistemática para guardar y restaurar el estado de cada página, sobre todo si no se está usando MVVM.

En una aplicación MVVM, generalmente un modelo de vista es responsable de guardar los datos que subyace en las diferentes páginas de una aplicación. Sin embargo, en ausencia de un modelo de vista, ese trabajo se deja a cada página individual, por lo general implica la propiedades diccionario implementado por el `Solicitud` clase. Sin embargo, es necesario tener cuidado de no tener duplicados claves de diccionario en dos o más páginas. Duplicados de las llaves son particularmente probable si un tipo de página en particular puede tener varias instancias de la pila de navegación.

El problema de las claves del diccionario duplicados puede evitarse si cada página en la pila de navegación utiliza un prefijo único para sus claves de diccionario. Por ejemplo, la página principal podría utilizar un prefijo de "0" para todas sus claves de diccionario, la página siguiente en la pila de navegación podría utilizar el prefijo "1" y así sucesivamente.

los `Xamarin.FormsBook.Toolkit` biblioteca tiene una interfaz y una clase que trabajan juntos para ayudarle con guardar y restaurar la pila de navegación, y guardar y restaurar el estado de página utilizando el diccionario únicos prefijos clave. Esta interfaz y la clase no excluyen el uso de MVVM con su solicitud.

La interfaz se llama `IPersistentPage`, y se ha nombrado métodos `Salvar` y `Restaurar` que incluyen el diccionario-tecla prefijo como argumento:

```
espacio de nombres Xamarin.FormsBook.Toolkit
{
    público interfaz IPersistentPage
```

```

    {
        vacio Salvar( cuerda prefijo);

        vacio Restaurar( cuerda prefijo);
    }
}

```

Cualquier página de la aplicación puede implementar IPersistentPage. los Salvar y Restaurar métodos son responsables de la utilización de la prefijo parámetro cuando la adición de los artículos a la propiedades diccionario o acceder a dichos elementos. Vas a ver ejemplos en breve.

Estas Salvar y Restaurar métodos se llaman de una clase llamada MultiPageRestorableApp, que se deriva de Solicitud y está destinado a ser una clase base para la Aplicación clase. Al derivar Aplicación de MultiPageRestorableApp, usted tiene dos responsabilidades:

- Desde el Aplicación El constructor de la clase, llame al Puesta en marcha método de MultiPageRestorableApp con el tipo de página principal de la aplicación.
- Llame a la base de la clase de OnSleep método de la OnSleep anulación de la Aplicación clase.

También hay dos requisitos al utilizar MultiPageRestorableApp:

- Cada página de la aplicación debe tener un constructor sin parámetros.
- Al derivar Aplicación de MultiPageRestorableApp, esta clase de base se convierte en un tipo de público expuesto desde la biblioteca de clases portátil de la aplicación. Esto significa que todos los proyectos de plataforma individuales también requieren una referencia a la **Xamarin.FormsBook.Toolkit** biblioteca.

MultiPageRestorableApp implementa su OnSleep método por el bucle a través de los contenidos de NavigationStack y ModalStack. Cada página se da un índice único a partir de 0, y cada página se reduce a una corta cadena que incluye el tipo de página, el índice de la página, y un valor booleano que indica si la página es modal:

```

espacio de nombres Xamarin.FormsBook.Toolkit
{
    // Las clases derivadas deben llamar puesta en marcha (typeof (YourStartPage));
    // Las clases derivadas deben llamar base.OnSleep () en override
    clase pública MultiPageRestorableApp : Solicitud
    {

        ...
        protegido override void OnSleep ()
        {
            StringBuilder pila de paginas = nuevo StringBuilder ();
            En t index = 0;

            // acumular los modales en las páginas pila de paginas.
            IReadonlyList < Página > Pila = ( MainPage como NavigationPage ) .Navigation.NavigationStack;
            LoopThroughStack (pila de paginas, pila, árbito Índice, falso );

            // acumular los modales en las páginas pila de paginas.
            pila = ( MainPage como NavigationPage ) .Navigation.ModalStack;
            LoopThroughStack (pila de paginas, pila, árbito Índice, cierto );
        }
    }
}

```

Además, cada página que implementa IPersistentPage recibe una llamada a su Salvar método con el prefijo número entero convierte en una cadena.

Los OnSleep métodos concluyen por el ahorro de la cadena compuesta que contiene una línea por cada página a las propiedades diccionario con la tecla "pila de páginas".

Un Aplicación clase que deriva de MultiPageRestorableApp debe llamar a la Puesta en marcha método de su constructor. Los Puesta en marcha método accede a la entrada "pila de páginas" en el propiedades diccionario. Para cada línea, se crea la instancia de una página de ese tipo. Si los implementos página IPersistentPage, entonces el Re-almacenar método se llama. Cada página se agrega a la pila de navegación con una llamada a PushAsync o Empujar-ModalAsync. Note el segundo argumento PushAsync y PushModalAsync se establece en falso para suprimir cualquier animación página a la transición de la plataforma podría poner en práctica:

```
espacio de nombres Xamarin.FormsBook.Toolkit

{
    // Las clases derivadas deben llamar Puesta en marcha (typeof(YourStartPage));
    // Las clases derivadas deben llamar base.OnSleep () en override
    clase publica MultiPageRestorableApp : Solicitud

    {
        protected void Puesta en marcha( Tipo startPageType)
        {
            objeto valor;
            Si (Properties.TryGetValue ( "Pila de paginas" , fuera valor))

```

```
{  
    MainPage = nuevo NavigationPage();  
    RestorePageStack (( cuerda )valor);  
}  
más  
{  
    // La primera vez que se ejecuta el programa.  
    Montaje montaje = esta .GetType () GetTypeInfo () Asamblea.;  
    Página page = ( Página ) Activador.CreateInstance (startPageType);  
    MainPage = nuevo NavigationPage (página);  
}  
}  
  
vacío asíncrono RestorePageStack ( cuerda pila de páginas)  
{  
    Montaje montaje = GetType () GetTypeInfo () Asamblea.;  
    StringReader lector = nuevo StringReader (Pila de páginas);  
    cuerda línea = nulo ;  
  
    // Cada línea es una página en la pila de navegación.  
    mientras ( nulo != (Línea = reader.ReadLine ()) )  
    {  
        cuerda [] Dividida = línea.split ( " ");  
        cuerda pageTypeName = split [0];  
        cuerda prefijo = split [1] + " ";  
        bool IsModal = Boole.Parse (split [2]);  
  
        // instancia de la página.  
        Tipo quiénes somos? = assembly.GetType (pageTypeName);  
        Página page = ( Página ) Activador.CreateInstance (quiénes somos?);  
  
        // Llamar Restaurar en la página si está disponible.  
        Si (página es IPersistentPage )  
        {  
            (( IPersistentPage ) Página ) .Restore (prefijo);  
        }  
  
        Si (! IsModal)  
        {  
            // Vaya a la página siguiente no modal.  
            esperar MainPage.Navigation.PushAsync (página, falso );  
  
            // Hack: para permitir la navegación por la página para completar!  
            Si ( Dispositivo .os == TargetPlatform .Windows &&  
                Dispositivo .Idiom != TargetIdiom .Teléfono)  
                esperar Tarea .Delay (250);  
        }  
        más  
{  
            // Vaya a la página siguiente modal.  
            esperar MainPage.Navigation.PushModalAsync (página, falso );  
  
            // Hack: para permitir la navegación por la página para completar!  
            Si ( Dispositivo .os == TargetPlatform .iOS)
```

```
        esperar Tarea .Delay (100);  
    }  
}  
}  
}  
...  
}  
}  
}
```

Este código contiene dos comentarios que comienzan con la palabra "Hack". Estos indican los estados que tienen la intención de solucionar dos problemas encontrados en Xamarin.Forms:

- En iOS, páginas modales anidados no restauran correctamente a menos que un poco de tiempo separa el PushModalAsync llamadas.
 - En Windows 8.1, páginas modales no contienen flecha hacia la izquierda **Espalda** botones menos un poco de tiempo separa las llamadas a PushAsync.

Vamos a probarlo!

los **StackRestoreDemo** programa tiene tres páginas, llamado **DemoMainPage**, **DemoModelessPage**, y **DemoModalPage**, cada uno de los cuales contiene una paso a paso e implementos **IPersistentPage** para guardar y restaurar la **Valor** propiedad asociada con ese Paso a paso. Se pueden establecer diferentes paso a paso Los valores en cada página y después comprobar si están restaurados correctamente.

los Aplicación clase se deriva de MultiPageRestorableApp. se llama Puesta en marcha de su constructor y llama a la clase base OnSleep método de su OnSleep anular:

```
clase pública Aplicación : Xamarin.FormsBook.Toolkit. MultiPageRestorableApp
{
    público App ()
    {
        // Debe llamar Puesta en marcha con el tipo de página de inicio!
        Puesta en marcha( tipo de ( Demo MainPage ) );
    }

    protegido override void OnSleep ()
    {
        // Debe llamar implementación base!
        base .OnSleep ();
    }
}
```

El XAML para Demo MainPage instancia un paso a paso, un Etiqueta mostrando el valor de ese paso a paso, y dos Botón elementos:

```
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
      xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml
      x: Class = " StackRestoreDemo.DemoMainPage "
      Titulo = " Pagina principal " >

< StackLayout >
    < Etiqueta Texto = " Pagina principal " >
```

```

    Tamaño de fuente = " Grande "
    VerticalOptions = " CenterAndExpand "
    HorizontalOptions = " Centrar " />

    < Cuadrícula VerticalOptions = " CenterAndExpand " >
        < peso a peso x: Nombre = " peso a peso "
            Grid.Column = " 0 "
            VerticalOptions = " Centrar "
            HorizontalOptions = " Centrar " />

        < Etiqueta Grid.Column = " 1 "
            Texto = "(Binding Fuente = {x: peso a peso Referencia}),
                    Path = Valor,
                    StringFormat = '{0: F0}')"
            Tamaño de fuente = " Grande "
            VerticalOptions = " Centrar "
            HorizontalOptions = " Centrar " />
    </ Cuadrícula >

    < Botón Texto = " Ir a la Página no modal "
        Tamaño de fuente = " Grande "
        VerticalOptions = " CenterAndExpand "
        HorizontalOptions = " Centrar "
        hecho clic = " OnGoToModelessPageClicked " />

    < Botón Texto = " Ir a la página modal "
        Tamaño de fuente = " Grande "
        VerticalOptions = " CenterAndExpand "
        HorizontalOptions = " Centrar "
        hecho clic = " OnGoToModalPageClicked " />

</ StackLayout >
</ Página de contenido >
```

Los controladores de eventos para los dos Botón elementos navegar hasta DemoModelessPage y DemoModal-Página. La implementación de IPersistentPage Guarda y restaura la Valor propiedad de la Paso- por elemento mediante el uso de la propiedades diccionario. Observe el uso de la prefijo parámetro para definir la clave de diccionario:

```

público clase parcial Demo MainPage : Página de contenido , IPersistentPage
{
    público Demo MainPage ()
    {
        InitializeComponent ();
    }

    vacío asíncrono OnGoToModelessPageClicked ( objeto remitente, EventArgs args)
    {
        esperar Navigation.PushAsync ( nuevo DemoModelessPage ());
    }

    vacío asíncrono OnGoToModalPageClicked ( objeto remitente, EventArgs args)
    {
        esperar Navigation.PushModalAsync ( nuevo DemoModalPage ());
    }
}
```

```

}

public void Salvar( cuerda prefijo)
{
    Aplicación .Current.Properties [prefijo + "StepperValue"] = Stepper.Value;
}

public void Restaurar( cuerda prefijo)
{
    objeto valor;
    Si (Aplicación .Current.Properties.TryGetValue (prefijo + "StepperValue", fuera valor))
        stepper.Value = ( doble )valor;
}
}

```

los DemoModelessPage clase es esencialmente el mismo que DemoMainPage excepto por el Título propiedad y el Etiqueta que muestra el mismo texto que la Título.

los DemoModalPage es algo diferente. También tiene una paso a paso y una Etiqueta que muestra el valor de la paso a paso, pero uno Botón regresa a la página anterior y la otra Botón navega a otra página modal:

```

< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " StackRestoreDemo.DemoModalPage "
    Título = " Página modal " >

< StackLayout >
    < Etiqueta Texto = " Página modal " *
        Tamaño de fuente = " Grande "
        VerticalOptions = " CenterAndExpand "
        HorizontalOptions = " Centrar " />

    < Cuadricula VerticalOptions = " CenterAndExpand " >
        < paso a paso x: Nombre = " paso a paso "
            Grid.Column = " 0 "
            VerticalOptions = " Centrar "
            HorizontalOptions = " Centrar " />

        < Etiqueta Grid.Column = " 1 "
            Texto = " {Binding Fuente = {x: paso a paso Referencia},
                    Path = Valor,
                    StringFormat = '{0: F0}' } "
            Tamaño de fuente = " Grande "
            VerticalOptions = " Centrar "
            HorizontalOptions = " Centrar " />
    
```

</ Cuadricula >

```

    < Botón Texto = " Regresa "
        Tamaño de fuente = " Grande "
        VerticalOptions = " CenterAndExpand "
        HorizontalOptions = " Centrar "
        hecho clic = " OnGoBackClicked " />

```

```

< Botón x: Nombre = "gotoModalButton" 
    Texto = "Ir a la página modal "
    Tamaño de fuente = "Grande "
    VerticalOptions = "CenterAndExpand "
    HorizontalOptions = "Centrar "
    hecho clic = "OnGoToModalPageClicked " />

</ StackLayout >
</ Página de contenido >

```

El archivo de código subyacente contiene controladores para los dos botones y también implementa IPersistentPage:

```

pública clase parcial DemoModalPage : Página de contenido , IPersistentPage
{
    pública DemoModalPage ()
    {
        InitializeComponent ();
    }

    vacío asíncrono OnGoBackClicked ( objeto remitente, EventArgs args)
    {
        esperar Navigation.PopModalAsync ();
    }

    vacío asíncrono OnGoToModalPageClicked ( objeto remitente, EventArgs args)
    {
        esperar Navigation.PushModalAsync ( nuevo DemoModalPage ());
    }

    public void Salvar( cuerda prefijo)
    {
        Aplicación .Current.Properties [prefijo + "StepperValue"] = Stepper.Value;
    }

    public void Restaurar( cuerda prefijo)
    {
        objeto valor;
        Si ( Aplicación .Current.Properties.TryGetValue (prefijo + "StepperValue" , fuera valor))
            stepper.Value = ( doble )valor;
    }
}

```

Una manera fácil de probar el programa es para navegar progresivamente a varias páginas no modales y modales, establecer un valor diferente en el paso a paso en cada página. A continuación, terminar la aplicación desde el teléfono o emulador (como se describió anteriormente) y empezar de nuevo. Usted debe estar en la misma página que la página que a la izquierda y ver el mismo paso a paso los valores a medida que avanza a través de las páginas.

Algo así como una aplicación en la vida real

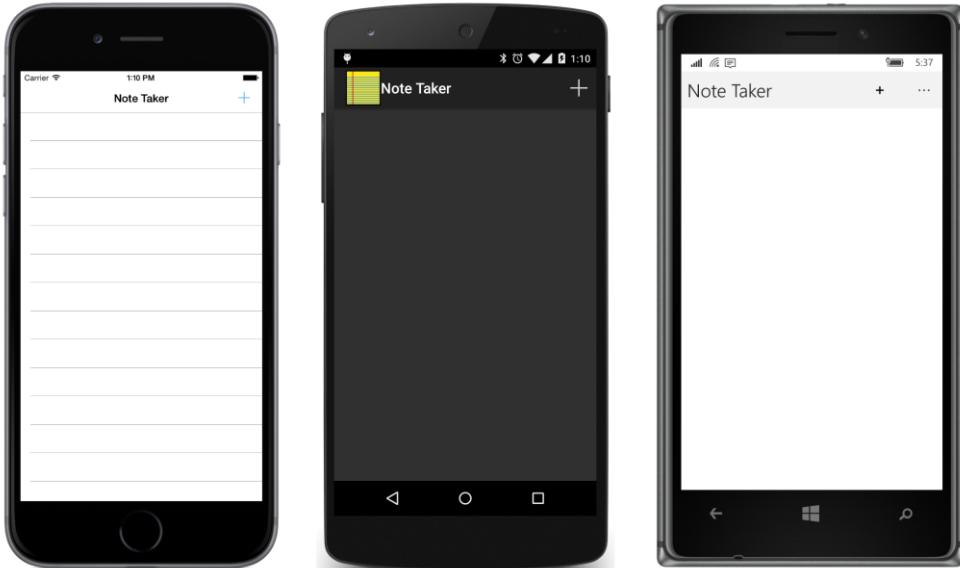
Idealmente, los usuarios no deben tener en cuenta cuando una aplicación se termina y se reinicia. La experiencia de la aplicación debe ser continua y sin interrupciones. Una media introducida Entrada que nunca fue completada debe

todavía estar en el mismo estado una semana más tarde, incluso si el programa no ha estado funcionando durante todo ese tiempo.

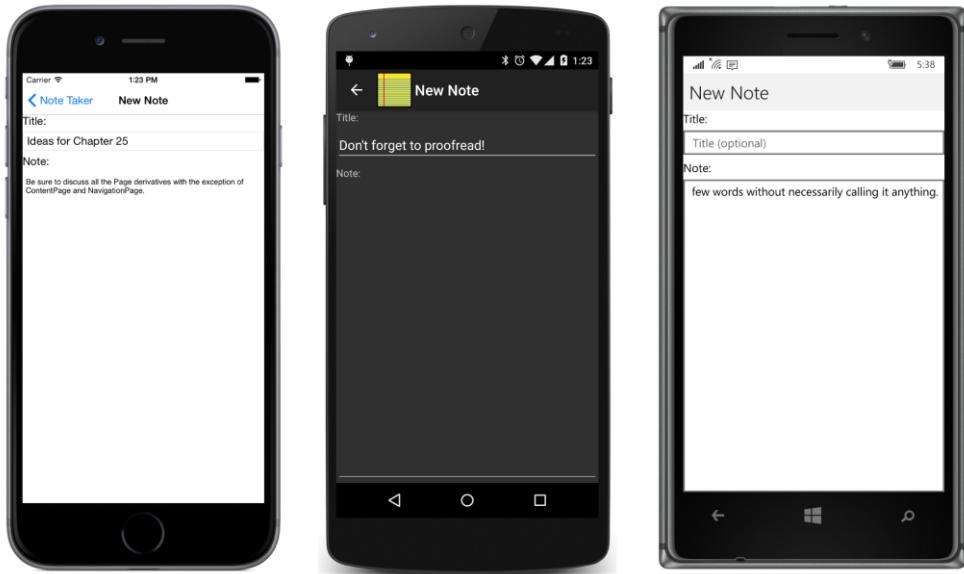
los **Tomador de notas** programa permite al usuario tomar notas que consisten en un título y un texto. Debido a que puede haber un buen número de estas notas, y tienen el potencial de convertirse en bastante largo, no se almacenan en el propiedades diccionario. En su lugar, el programa hace uso de la **IFileHelper** interfaz y **FileHelper** clases demostradas en el **TextFileAsync** programa en el capítulo 20, “asíncrono y el archivo de E / S.” Cada nota es un archivo independiente. Me gusta **TextFileAsync**, el **Tomador de notas** solución también contiene todos los proyectos en el **Xamarin.FormsBook.Platform** solución y referencias a los proyectos de biblioteca.

Tomador de notas se estructura muy similar a la Transferencia de datos programas anteriores en este capítulo, con las páginas nombradas **NoteTakerHomePage** y **NoteTakerNotePage**.

La página principal se compone de una **ItemsView** que domina la página y una **Añadir** botón. Esta **Añadir** es un botón **ToolbarItem** que toma la forma de un signo más en la esquina superior derecha de las pantallas de iOS, Android y Windows Phone:

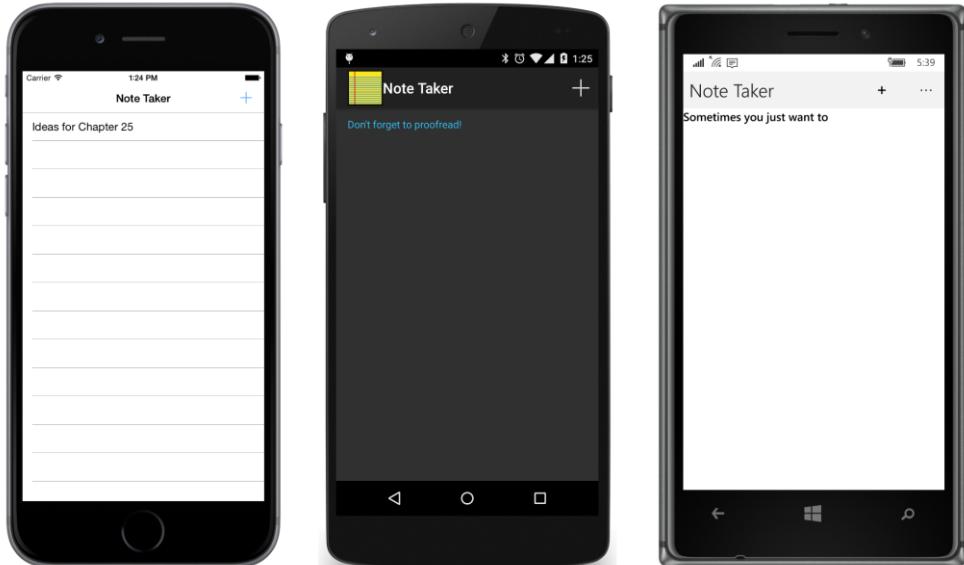


Al pulsar ese botón hace que el programa vaya a la **NoteTakerNotePage**. En la parte superior es una Entrada por un título, pero la mayor parte de la página está ocupada por una **Editor** El texto de la nota misma. Ahora puede escribir un título y una nota:

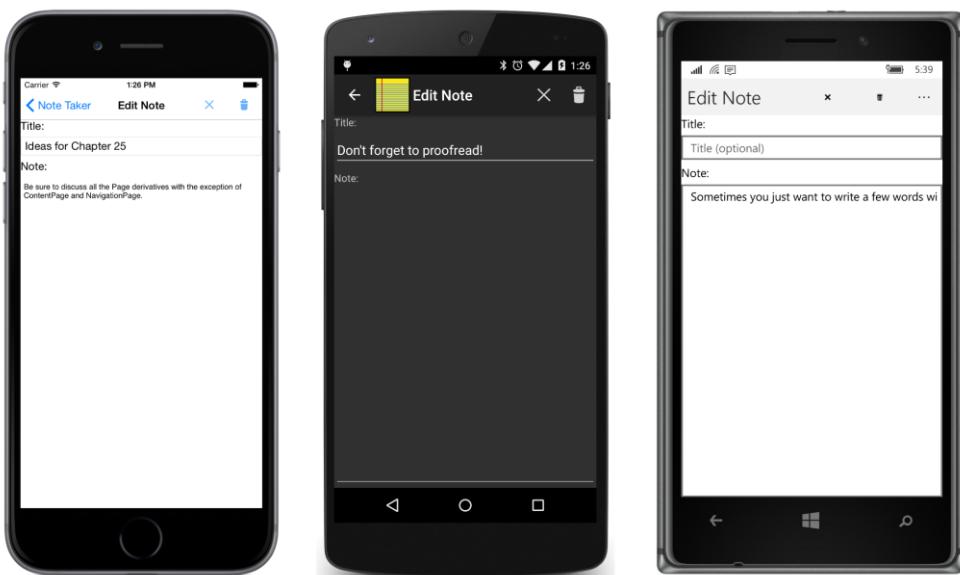


No es necesario introducir un título. Si no lo hay, un identificador se construye que consiste en el principio del texto. Tampoco es necesario para entrar en el texto de la nota. Una nota puede consistir únicamente de un título. (En el momento de escribir este capítulo, el tiempo de ejecución de Windows Editor no envuelva correctamente el texto.)

Si el título o nota no está en blanco, la nota se considera que es una nota válida. Cuando vuelva a la página principal mediante el uso de la norma **Espalda** cualquiera de los botones en la barra de navegación o en la parte inferior de la pantalla, se añade la nueva nota a la Vista de la lista:



Ahora puede añadir más nuevas notas o editar una nota existente pulsando en la entrada en el Vista de la lista. Los Vista de la lista grifo navega a la misma página que el Añadir botón, pero notar que la Título propiedad en la segunda página es ahora "Editar nota" en lugar de "Nueva nota":



Ahora puede realizar cambios en la nota y volver volver a la página principal. Dos elementos de la barra de herramientas también están disponibles en esta página: La primera es una Cancelar botón que le permite abandonar cualquier edición que haya realizado. Una alerta le pregunta si está seguro. También puede tocar el Borrar elemento para borrar la nota, también con una alerta de confirmación.

Uno de los aspectos difíciles de este programa consiste en la Cancelar botón. Supongamos que estás en el medio de la edición de una nota y te distraes, y, finalmente, se termina el programa. La próxima vez que inicie el programa, se debe volver a la pantalla de edición y ver los cambios. Si a continuación invoca el Cancelar mando, las modificaciones debe ser abandonada.

Esto significa que cuando la aplicación se suspende mientras una nota está siendo editado, la nota debe esencialmente ser salvado en dos formas diferentes: El pre-editar nota y la nota con las ediciones. El programa se encarga de esta por el ahorro de cada nota en un archivo sólo cuando NoteTakerNotePage recibe una llamada a su OnDisappearing anular. (Sin embargo, alguna consideración especial necesita para acomodar el caso de iOS cuando la página recibe una llamada a OnDisappearing cuando el programa termina.) La versión del archivo de la Nota objeto es el uno sin las modificaciones. La versión editada se refleja en el contenido actual de la Entrada y Editor; NoteTakerNotePage salva a los dos cadenas de texto en el Salvar método de su IPersistentPage implementación.

Los Nota implementa la clase INotifyPropertyChanged en virtud de que deriva de ViewModelBase en el Xamarin.FormsBook.Toolkit biblioteca. La clase define cuatro propiedades públicas: Nombre de archivo, título, Texto, y identificador, que o bien es el mismo que Título o los primeros 30 caracteres de Texto, truncado

para mostrar palabras completas solamente. Los Nombre del archivo propiedad se establece desde el constructor y nunca cambia:

```
público clase Nota : ViewModelBase , IEquatable < Nota >
{
    cuerda título, texto, identificador;
    FileHelper fileHelper = nuevo FileHelper ();

    público Nota( cuerda nombre del archivo)
    {
        Nombre = nombre de archivo;
    }

    público cuerda Nombre del archivo { conjunto privado ; obtener ; }

    público cuerda Título
    {
        conjunto
        {
            Si ( SetProperty ( árbito título, valor ))
            {
                Identificador = MakelIdentifier ();
            }
        }
        obtener { regreso título; }
    }

    público cuerda Texto
    {
        conjunto
        {
            Si ( SetProperty ( árbito texto, valor ) && Cuerda.IsNullOrWhiteSpace (título))
            {
                Identificador = MakelIdentifier ();
            }
        }
        obtener { regreso texto; }
    }

    público cuerda identificador
    {
        conjunto privado { SetProperty ( árbito identificador, valor ); }
        obtener { regreso identificador; }
    }

    cuerda MakelIdentifier ()
    {
        Si ( ! Cuerda.IsNullOrWhiteSpace ( esta .Título))
            regreso Título;

        En t truncationLength = 30;

        Si (Texto == nulo || Text.length <= truncationLength)
        {
            regreso Texto;
```

```
}

cuerda truncado = Text.Substring (0, truncationLength);

En t index = truncated.LastIndexOf ( " ");

Si (índice! = -1)
    truncado = truncated.Substring (0, index);

regreso truncado;
}

público Tarea SaveAsync ()
{
    cuerda text = nombre + Ambiente .NewLine + Texto;
    regreso fileHelper.WriteTextAsync (Nombre de archivo, de texto);
}

asíncrono pública Tarea LoadAsync ()
{
    cuerda text = esperar fileHelper.ReadTextAsync (filename);

    // Romper cadena en título y texto.

    En t index = text.IndexOf ( Ambiente .Nueva línea);
    Titulo = text.Substring (0, index);
    Text = text.Substring (indice + Ambiente .NewLine.Length);
}

asíncrono pública Tarea DeleteAsync ()
{
    esperar fileHelper.DeleteAsync (filename);
}

public bool es igual a ( Nota otro)
{
    regreso otros == nulo ? falso : Nombre de archivo == other.Filename;
}
}
```

los Nota clase también define métodos para guardar, cargar o eliminar el archivo asociado con la instancia particular de la clase.

La primera línea del archivo es la Título propiedad, y el resto del archivo es el Texto propiedad.

En la mayoría de los casos, hay un uno-a-uno correspondencia entre Nota los archivos y las instancias de la Nota clase. Sin embargo, si el DeleteAsync método se llama, entonces la Nota todavía existe objeto, pero el archivo no lo hace. (Sin embargo, como se verá, todas las referencias a una Nota objeto cuya DeleteAsync método que se llama son rápidamente independiente y el objeto se vuelve elegible para la recolección de basura.)

El programa no mantiene una lista de estos archivos cuando el programa no se está ejecutando. En cambio, el NoteFolder clase obtiene todos los archivos en el almacenamiento local de la aplicación con una extensión de nombre de ".Nota" y crea una colección de Nota objetos de estos archivos:

```

pública clase NoteFolder
{
    pública NoteFolder ()
    {
        esta .Notas = nuevo ObservableCollection < Nota > 0;
        GetFilesAsync ();
    }

    pública ObservableCollection < Nota > Notas { conjunto privado ; obtener ; }

    vacío asíncrono GetFilesAsync ()
    {
        FileHelper fileHelper = nuevo FileHelper ();

        // Clasificar los nombres de archivo.
        IEnumurable < cuerda > = nombres de archivo
            de nombre del archivo en lo esperan fileHelper.GetFilesAsync ()
            donde filename.EndsWith ( ".Nota" )
            OrdenarPor (nombre del archivo)
            seleccionar nombre del archivo;

        // almacenarlos en la colección Notas.
        para cada ( cuerda nombre del archivo en nombres de archivo)
        {
            Nota nota = nuevo Nota (nombre del archivo);
            esperar note.LoadAsync ();
            Notes.Add (nota);
        }
    }
}

```

Como verá, el nombre del archivo se construye a partir de una Fecha y hora oponerse en el momento de la nota se crea por primera vez, que consiste en el año seguido por el mes, el día y la hora, lo que significa que cuando éstos Nota los archivos se ordenan por nombre de archivo, que aparecen en la colección en el mismo orden en el que se crean.

los Aplicación crea una instancia de la clase NoteFolder y lo hace disponible como una propiedad pública. Aplicación deriva de MultiPageRestorableApp, por lo que llama Puesta en marcha con el NoteTakerHomePage escriba, y también implementa la OnSleep anular llamando a la implementación de la clase base:

```

pública clase Aplicación : MultiPageRestorableApp
{
    pública App ()
    {
        // Esta carga todos los archivos existentes .Nota.
        NoteFolder = nuevo NoteFolder ();

        // Hacer llamada al método en el MultiPageRestorableApp.
        Puesta en marcha( tipo de ( NoteTakerHomePage ));
    }

    pública NoteFolder NoteFolder { conjunto privado ; obtener ; }

    protegido override void OnSleep ()

```

```

{
    // Llamada requiere cuando se deriva de MultiPageRestorableApp.
    base.OnSleep();
}

// Procesamiento especial para iOS.
protected override void OnResumen()
{
    NoteTakerNotePage NotePage =
        ((NavigationPage)MainPage).CurrentPage as NoteTakerNotePage;

    Si (NotePage != null)
        notePage.OnResume();
}
}

```

los Aplicación clase también anula la En resumen método. Si NoteTakerNotePage esta actualmente activo, el método llama a una En resumen método en la página de notas. Se trata de un procesamiento especial para iOS. Como se verá, NoteTakerNotePage ahora Nota oponerse a un archivo durante su OnDisappearing anula, pero no debe hacerlo si el OnDisappearing override indica que la aplicación está finalizando.

El archivo XAML para el NoteTakerHomePage crea una instancia del Vista de la lista para la visualización de toda la Nota objetos. Los ItemsSource está unido a la notas colección de la NoteFolder que se almacena en el Aplicación clase. Cada Nota objeto se muestra en la Vista de la lista con su identificador propiedad:

```

< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " NoteTaker.NoteTakerHomePage "
    Título = " Tomador de notas " >

    < Vista de la lista ItemsSource = " {Binding Fuente = {x: Estático Application.Current},
        Path = NoteFolder.Notes} "
        selectedItem = " OnListViewItemSelected "
        VerticalOptions = " FillAndExpand " >
        < ListView.ItemTemplate >
            < DataTemplate >
                < TextCell Texto = " {Identificador Encuadernación} " />
            </ DataTemplate >
        </ ListView.ItemTemplate >
    </ Vista de la lista >

    < ContentPage.ToolbarItems >
        < ToolbarItem Nombre = " Añadir la nota "
            Orden = " Primario "
            Activado = " OnAddNoteActivated " >
            < ToolbarItem.Icon >
                < OnPlatform x: TypeArguments = " FileImageSource "
                    iOS = " new.png "
                    Androide = " ic.action_new.png "
                    WinPhone = " Imágenes / add.png " />
            </ ToolbarItem.Icon >
        </ ToolbarItem >
    </ ContentPage.ToolbarItems >

```

</ Página de contenido >

El archivo de código subyacente se dedica a la manipulación de tan sólo dos eventos: la itemSelected caso de la Vista de la lista para editar una ya existente Nota, y el Activado caso de la ToolbarItem para la creación de una nueva Nota:

```
parcial clase NoteTakerHomePage : Pagina de contenido
{
    público NoteTakerHomePage ()
    {
        InitializeComponent ();
    }

    vacío asíncrono OnListViewItemSelected ( objeto remitente, SelectedItemChangedEventArgs args)
    {
        Si (Args.SelectedItem = nulo )
        {
            // Desactive el artículo.
            Vista de la lista ListView = ( Vista de la lista )remitente;
            listView.SelectedItem = nulo ;

            // Vaya a NotePage.
            esperar Navigation.PushAsync ( nuevo NoteTakerNotePage
            {
                Nota = ( Nota ) Args.SelectedItem,
                IsNoteEdit = cierto
            });
        }
    }

    vacío asíncrono OnAddNoteActivated ( objeto remitente, EventArgs args)
    {
        // crear nombre de archivo único.
        Fecha y hora fechaHora = Fecha y hora .UtcNow;
        cuerda nombre de archivo = DateTime.ToString ( "yyyMMddHHmmssfff" ) + ".Nota" ;

        // Vaya a NotePage.
        esperar Navigation.PushAsync ( nuevo NoteTakerNotePage
        {
            Nota = nuevo Nota ( nombre del archivo),
            IsNoteEdit = falso
        });
    }
}
```

En ambos casos, el controlador de eventos instancia NoteTakerNotePage, establece dos propiedades, y navega a esa página. Para una nueva nota, un nombre de archivo se construye y una Nota objeto se crea una instancia. Para una nota existente, la Nota objeto es simplemente el elemento seleccionado de la Vista de la lista. Tenga en cuenta que la nueva nota tiene un nombre de archivo, pero aún no se guarda en un archivo o formar parte de las notas recogida en NoteFolder.

El archivo XAML para NoteTakerNotePage tiene una Entrada y Editor para la introducción de título y el texto de una nota. Los enlaces de datos sobre la Texto propiedades de estos elementos implican que la BindingContext para

la página es una Nota objeto:

```
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns:x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " NoteTaker.NoteTakerNotePage "
    Título = " Nueva nota " >

    < StackLayout >
        < Etiqueta Texto = " Título: " />

        < Entrada Texto = " {Título} Encuadernación "
            marcador de posición = " Título (opcional) " />

        < Etiqueta Texto = " Nota: " />

        < Editor Texto = " {Texto} Encuadernación "
            Teclado = " Texto "
            VerticalOptions = " FillAndExpand " />

    </ StackLayout >

    < ContentPage.ToolbarItems >
        < ToolbarItem Nombre = " Cancelar "
            Orden = " Primario "
            Activado = " OnCancelActivated " >
            < ToolbarItem.Icon >
                < OnPlatform x: TypeArguments = " FileImageSource "
                    iOS = " cancel.png "
                    Androide = " ic_action_cancel.png "
                    WinPhone = " Imágenes / cancel.png " />
            </ ToolbarItem.Icon >
        </ ToolbarItem >

        < ToolbarItem Nombre = " Borrar "
            Orden = " Primario "
            Activado = " onDeleteActivated " >
            < ToolbarItem.Icon >
                < OnPlatform x: TypeArguments = " FileImageSource "
                    iOS = " discard.png "
                    Androide = " ic_action_discard.png "
                    WinPhone = " Imágenes / delete.png " />
            </ ToolbarItem.Icon >
        </ ToolbarItem >
    </ ContentPage.ToolbarItems >
</ Pagina de contenido >
```

Los dos ToolbarItem elementos hacia la parte inferior debe ser visible sólo cuando se está editando una nota existente. La eliminación de estos elementos de la barra se produce en el archivo de código subyacente cuando el IsNoteEdit propiedad se establece desde la página principal. Ese código también cambia el Título propiedad de la página. los conjunto para el acceso Nota la propiedad es responsable de establecer la página de BindingContext:

```
publco clase parcial NoteTakerNotePage : Pagina de contenido , IPersistentPage
{
    Nota Nota;
    bool isNoteEdit;
```

```

...


```

los NoteTakerNotePage clase implementa la IPersistentPage interfaz, lo que significa que se ha llamado métodos Salvar y Restaurar para guardar y restaurar el estado de la página. Estos métodos utilizan la propiedades diccionario para guardar y restaurar las propiedades de los tres Nota que definen una Nota objeto la Nombre de archivo, título, y Texto propiedades y la IsNoteEdit propiedad de Tomador de notas-

NotePage. Este es el Nota oponerse en su estado actual editado:

```

público clase parcial NoteTakerNotePage : Página de contenido , IPersistentPage
{
    ...
    // campo especial para iOS.
    bool isInSleepState;
    ...

    // aplicación IPersistent
    public void Salvar( cuerda prefijo)
    {
        // código especial para iOS.
        isInSleepState = cierto ;

        Solicitud aplicación = Solicitud .Corriente;
    }
}

```

```

        app.Properties [ "nombre del archivo" ] = Note.Filename;
        app.Properties [ "título" ] = Note.Title;
        app.Properties [ "texto" ] = Note.Text;
        app.Properties [ "IsNoteEdit" ] = IsNoteEdit;
    }

    public void Restaurar( cuerda prefijo)
    {
        Solicitud aplicación = Solicitud .Corriente;

        // crear un nuevo objeto Nota.
        Nota nota = nuevo Nota (( cuerda ) app.Properties [ "nombre del archivo" ]);
        {

            Title = ( cuerda ) app.Properties [ "título" ],
            Text = ( cuerda ) app.Properties [ "texto" ]
        };

        // Establecer las propiedades de esta clase.
        Nota = nota;
        IsNoteEdit = ( bool ) app.Properties [ "IsNoteEdit" ];
    }

    // código especial para iOS.
    public void En resumen()
    {
        isInSleepState = falso ;
    }
    ...
}

```

La clase también define un método llamado `En resumen` que se llama desde el `Aplicación clase`. Por lo tanto, la `isInSleepState` campo es cierto cuando la aplicación se ha suspendido.

El propósito de `isInSleepState` campo es evitar guardar la Nota en un archivo cuando la `OnDisappearing` override se llama como la aplicación se termina bajo IOS. el ahorro de este Nota objeto a un archivo no permitiría que el usuario abandone tarde ediciones de la Nota pulsando la `Cancelar` botón en esta página.

Si el `OnDisappearing` anulación indica que el usuario está regresando de nuevo a la página de inicio, como lo hace de otra manera en esta solicitud, entonces el Nota objeto se puede guardar en un archivo, y, posiblemente, añadido a la notas recogida en `NoteFolder`:

```

público clase parcial NoteTakerNotePage : Página de contenido , IPersistentPage
{
    ...
    asíncrono protected void override OnDisappearing ()
    {
        base .OnDisappearing ();

        // código especial para iOS:
        //          No guardar la nota cuando el programa está terminando.
        Si (isInSleepState)
            regreso ;
    }
}

```

```

// guardar sólo la nota si hay algo de texto en alguna parte.
Si (! Cuerda.IsNullOrWhiteSpace (Nota.Title) ||
    ! Cuerda.IsNullOrWhiteSpace (Nota.Text))
{
    // Guarda la nota en un archivo.
    esperar Note.SaveAsync ();
}

// Añadir a la colección si se trata de una nota nueva.
NoteFolder noteFolder = ((Aplicación ) Aplicación .Current) .NoteFolder;

// método IndexOf encuentra partido basado en la propiedad Nombre del archivo
// basado en la aplicación de IEquatable en la nota.
En t index = noteFolder.Notes.IndexOf (nota);
Si (Indice == -1)
{
    // No hay resultados - añadirlo.
    noteFolder.Notes.Add (nota);
}
más
{
    // Partido - reemplazarlo.
    noteFolder.Notes [Indice] = Nota;
}
}

...
}
}

```

los Nota clase implementa la IEquatable interfaz y define dos Nota objetos que sean iguales si sus Nombre del archivo propiedades son las mismas. los OnDisappearing anulación se basa en que la definición de igualdad para evitar la adición de una Nota oponerse a la recogida si ya existe otro con el mismo Nombre del archivo propiedad.

Finalmente, el NoteTakerNotePage archivo de código subyacente tiene controladores para los dos ToolbarItem elementos. En ambos casos, el proceso comienza con una llamada a DisplayAlert para obtener la confirmación del usuario, y, o bien vuelve a cargar el Nota objeto desde el archivo (sobrescribiendo eficazmente cualquier edición), o borra el archivo y lo elimina de la notas colección:

```

público clase parcial NoteTakerNotePage : Página de contenido , IPersistentPage
{
    ...
    vacío asíncrono OnCancelActivated ( objeto remitente, EventArgs args)
    {
        Si (esperar DisplayAlert ("Tomador de notas" , "Cancelar nota de edición?" ,
            "Sí" , "No" ))
        {
            // Actualizar nota.
            esperar Note.LoadAsync ();
        }

        // Regresar a la página principal.
        esperar Navigation.PopAsync ();
    }
}

```

```
}

vacío asíncrono OnDeleteActivated ( objeto remitente, EventArgs args)
{
    Si ( esperar DisplayAlert ( "Tomador de notas", "Eliminar esta nota?", "Sí", "No" ))
    {
        // Borrar el archivo de notas y quitar de la colección.
        esperar Note.DeleteAsync ();
        (( Aplicación ) Aplicación .Current) .NoteFolder.Notes.Remove (Nota);

        // Acabar con la entrada y por lo que el editor Nota
        // no se guardarán durante OnDisappearing.
        Note.Title = "";
        Note.Text = "";

        // Regresar a la página principal.
        esperar Navigation.PopAsync ();
    }
}
```

Por supuesto, esta no es la única manera de escribir un programa como este. Es posible mover una gran cantidad de la lógica para crear, editar y eliminar notas en Datos de aplicación y convertirlo en un modelo de vista adecuado. Datos de aplicación probablemente necesitará una nueva propiedad de tipo Nota llamado CurrentNote, y varias propiedades de tipo Yo ordeno para la unión a la Mando propiedad de cada una de las ToolbarItem elementos.

Algunos programadores incluso tratan de mover la lógica de página de navegación en ViewModels, pero no todo el mundo está de acuerdo en que este es un enfoque adecuado para MVVM. Es una parte de la página de la interfaz de usuario y por lo tanto parte de la vista? O son páginas realmente más como una colección de elementos de datos relacionados?

cuestiones filosóficas como las que podría llegar a ser aún más irritante como las variedades de tipos de página en Xamarin.Forms se exploran en el capítulo siguiente.

capítulo 25

variedades página

Si se piensa en una aplicación Xamarin.Forms como un edificio, a continuación, que la construcción de este edificio de ladrillos que toman la forma de puntos de vista y elementos. Usted los arreglos en las paredes utilizando las clases de diseño, y luego formarlos en habitaciones con Pagina de contenido, con pasajes de una habitación a que son posibles con las funciones de navegación estructuradas en torno NavigationPage.

Esta arquitectura visual se puede mejorar un poco más con otras clases que se derivan de instanciables Página. Aquí está la jerarquía completa:

```
Página
  TemplatedPage
    Pagina de contenido
  NavigationPage
  MasterDetailPage
  MultiPage <T>
    TabbedPage
    CarouselPage
```

Este capítulo está dedicado a estos adicionales Página derivados, que son similares en cuanto que sirven como padres para gestionar la presentación visual de dos o más páginas:

- **MasterDetailPage gestiona dos páginas:** La *dominio* en general es una colección de datos o una lista de elementos, y la *detalle* generalmente muestra un artículo particular de la colección.
- **TabbedPage se compone de múltiples páginas secundarias identificadas por pestañas.** Puede llenar TabbedPage con una colección de páginas discretas o automáticamente generar pestañas y las páginas en base a un conjunto de datos de la misma manera que una Vista de la lista genera elementos en función de un conjunto de datos. Con esta segunda opción, cada pestaña está asociada con un miembro de la colección, formateado con una plantilla, pero esta opción no es adecuada para las plataformas iOS.

los CarouselPage que verá la desaprobación a favor de una próxima CarouselView, por lo que no será discutido en este capítulo. MultiPage <T> es abstracta y no puede ser instanciado en sí, sino que define la mayoría de las propiedades y los eventos de TabbedPage.

Maestra y detalle

los MasterDetailPage define dos propiedades, nombrados Dominar y Detalle de tipo Página. En general, deberá definir estas dos propiedades a objetos de tipo Pagina de contenido, pero en la actualidad, para conseguir MasterDetail-Página para trabajar en la plataforma Windows universal, la página de detalle debe ser una NavigationPage.

Cómo MasterDetailPage pantallas y alterna entre estas dos páginas depende de varios factores: el sistema operativo subyacente, si se está ejecutando el programa en un teléfono o tableta, la orientación vertical u horizontal del dispositivo, y el establecimiento de una propiedad de MasterDetail-

Página llamado **MasterBehavior**. Varios comportamientos son posibles:

- *división*: Las páginas maestra y detalle se muestran al lado del otro, el maestro de la izquierda y el detalle de la derecha.
- *popover*: La página de detalles está animado para cubrir o cubrir parcialmente, la página maestra. Hay tres posibilidades:
 - o *diapositiva*: El detalle y la página maestra corredera hacia atrás y hacia adelante.
 - o *superposición*: La página de detalles cubre parcialmente la página maestra.
 - o *intercambiar*: La página de detalles oscurece por completo la página maestra.

En teoría, el **MasterBehavior** propiedad de MasterDetailPage le permite elegir entre el *división* y *popover* comportamientos. Establece esta propiedad a uno de los cinco miembros de la **MasterBehavior** enumeración:

- Defecto
- División
- SplitOnLandscape
- SplitOnPortrait
- popover

Como verá, sin embargo, el ajuste de la **MasterBehavior** propiedad tiene *sin efecto* para aplicaciones que se ejecutan en los teléfonos. Sólo afecta a las aplicaciones que se ejecutan en una tableta o el escritorio. Los teléfonos siempre exhiben una *popover* comportamiento. Si este comportamiento se traduce en una *diapositiva*, *se superponen*, o *intercambiar* depende de la plataforma.

La exploración de los comportamientos

Vamos a explorar estos comportamientos con un programa llamado **MasterDetailBehaviors**. El programa define tres páginas, llamado **DemoPage** (que se deriva de **MasterDetailPage**), y dos Página de contenido derivados que son hijos de la **MasterDetailPage**. Estos se denominan **Página principal** y **Detalle**-

Página.

Página principal y detalle son muy similares. A continuación se Página principal:

```
<Pagina de contenido xmlns = "http://xamarin.com/schemas/2014/forms"
    xmlns: x = "http://schemas.microsoft.com/winfx/2009/xaml"
    x: Class = "MasterDetailBehaviors.MasterPage"
    Titulo = "Página principal"
    Relleno = "10"
    x: Nombre = "página principal" >
```

```

< ContentPage.Padding >
    < OnPlatform x: TypeArguments = "Espesor" 
        iOS = "0, 20, 0, 0" />
</ ContentPage.Padding >

< Marco OutlineColor = "Acento" >
    < StackLayout Orientación = "Horizontal" 
        Espaciado = "0" 
        HorizontalOptions = "Centrar" 
        VerticalOptions = "Centrar" >

        < Etiqueta Texto = "{Binding Fuente = {x: masterpage Referencia}}, 
            Path = Ancho, 
            StringFormat = 'Maestro: {0: F0}'" 
            Tamaño de fuente = "Grande" />

        < Etiqueta Texto = "{Binding Fuente = {x: masterpage Referencia}}, 
            Path = Altura, 
            StringFormat = "& #X00D7; {0: F0}" 
            Tamaño de fuente = "Grande" />
    </ StackLayout >
</ Marco >
</ Página de contenido >

```

Contiene una Marco con un par de Etiqueta elementos para mostrar la anchura y la altura de la página. Tenga en cuenta que una Título propiedad se establece y que la página contiene la norma Relleno para evitar la superposición de la barra de estado en el iPhone.

los detalle que no contiene Relleno. Verá que es innecesario. Pero al igual que Dominar-Página, Esta página también establece el Título propiedad y contiene una Marco con un par de Etiqueta elementos para mostrar la anchura y altura:

```

< Página de contenido xmlns = "http://xamarin.com/schemas/2014/forms" 
    xmlns: x = "http://schemas.microsoft.com/winfx/2009/xaml" 
    x: Class = "MasterDetailBehaviors.DetailPage" 
    Título = "Página detalle" 
    Relleno = "10" 
    x: Nombre = "detalle" >

    < Marco OutlineColor = "Acento" >
        < StackLayout Orientación = "Horizontal" 
            Espaciado = "0" 
            VerticalOptions = "CenterAndExpand" 
            HorizontalOptions = "Centrar" >

            < Etiqueta Texto = "{Binding Fuente = {x: detalle de referencia}}, 
                Path = Ancho, 
                StringFormat = 'Detalle: {0: F0}'" 
                Tamaño de fuente = "Grande" />

            < Etiqueta Texto = "{Binding Fuente = {x: detalle de referencia}}, 
                Path = Altura, 
                StringFormat = "& #X00D7; {0: F0}" 
                Tamaño de fuente = "Grande" />
    </ StackLayout >
</ Marco >
</ Página de contenido >

```

```

</ StackLayout >
</ Marco >
</ Pagina de contenido >
```

Usted también necesitará una página que se deriva de MasterDetailPage. Para añadir una página de este tipo en Visual Studio, añadir un nuevo elemento al proyecto mediante el uso de la **Página forma Xaml** modelo; en Xamarin de estudio, añadir un nuevo archivo al proyecto mediante el uso de la **ContentPage forma Xaml** modelo. Esto crea una página que se deriva de Pagina de contenido, pero se puede entonces simplemente cambiar Pagina de contenido a MasterDetailPage tanto en el archivo XAML y C # archivo de código subyacente.

Aquí está el archivo XAML para DemoPage con MasterDetailPage como elemento raíz:

```

< MasterDetailPage xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    xmlns: locales = " CLR-espacio de nombres: MasterDetailBehaviors "
    x: Class = " MasterDetailBehaviors.DemoPage "
    Titulo = " Página de demostración "
    MasterBehavior = " Defecto " >

    < MasterDetailPage.Master >
        < locales: MasterPage />
    </ MasterDetailPage.Master >

    < MasterDetailPage.Detail >
        < NavigationPage >
            < x: Argumentos >
                < locales: detalle />
            </ x: Argumentos >
        </ NavigationPage >
    </ MasterDetailPage.Detail >
</ MasterDetailPage >
```

los **MasterDetailPage.Master** y **MasterDetailPage.Detail** elementos de propiedad se establecen en los casos de Página principal y de detalle, respectivamente pero con una pequeña diferencia: El **Detalle** propiedad se establece una **NavigationPage**, y el **x: Argumentos** etiquetas especifican el detalle como el argumento del constructor. Esto es necesario para que la interfaz de usuario que permite el cambio de usuario entre las páginas maestra y detalle en la plataforma Windows universal.

Observe también que la **MasterBehavior** propiedad se establece en **Defecto** en la etiqueta de raíz. Usted puede experimentar con diferentes configuraciones.

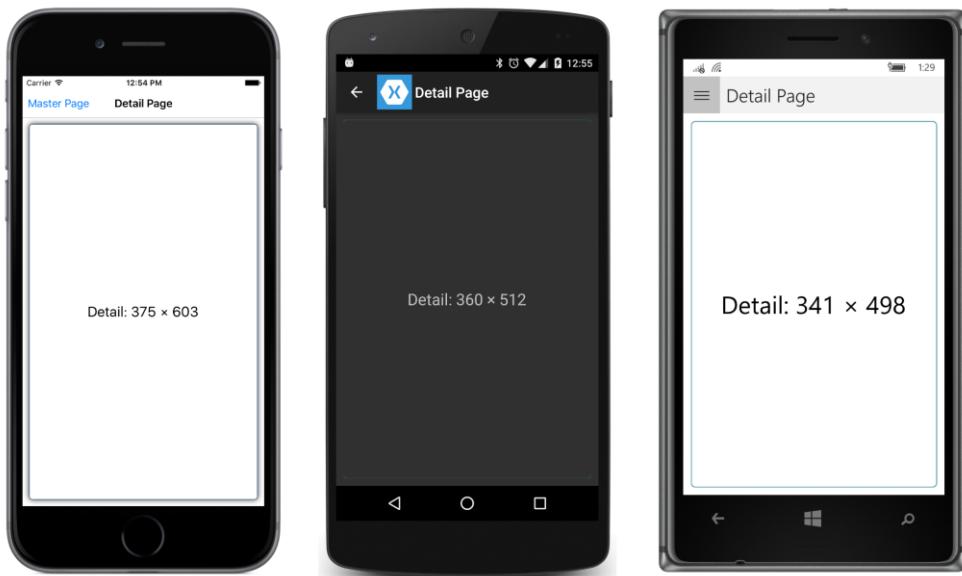
los Aplicación constructor establece la **Página principal** propiedad a **DemoPage**. Un programa Xamarin.Forms no debe desplazarse a una **MasterDetailPage**:

```

espacio de nombres MasterDetailBehaviors
{
    clase pública Aplicación : Solicitud
    {
        público App ()
        {
            MainPage = nuevo DemoPage ();
        }
    }
}
```

```
...  
}  
}
```

La primera vez que ejecute el programa, por defecto se muestra inicialmente la página de detalles:

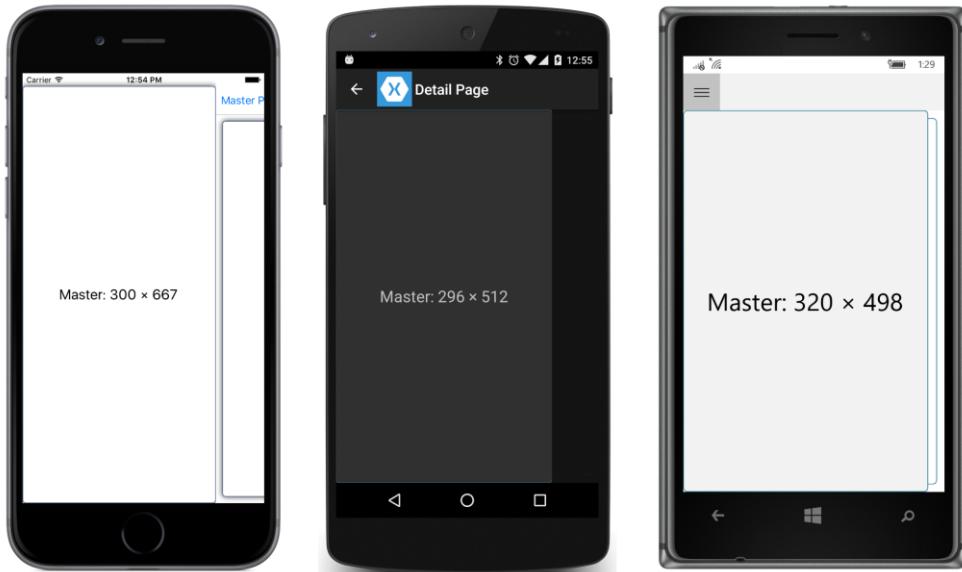


En las tres plataformas, el encabezamiento identifica esto como la página de detalles mediante la visualización del Título propiedad de De detalle. El iPhone también muestra el Título del Página principal.

La operación para cambiar de la página de detalle a la maestra es diferente en las tres plataformas:

- En iOS, deslizar la página de detalle a la derecha, o pulse el texto de páginas principales en el encabezamiento.
- En Android, desliza el dedo desde el borde izquierdo del teléfono, o toque la flecha en la esquina superior izquierda.
- En Windows 10 móvil, pulse el ícono de menú en la esquina superior izquierda.

Aquí está el resultado después del cambio:



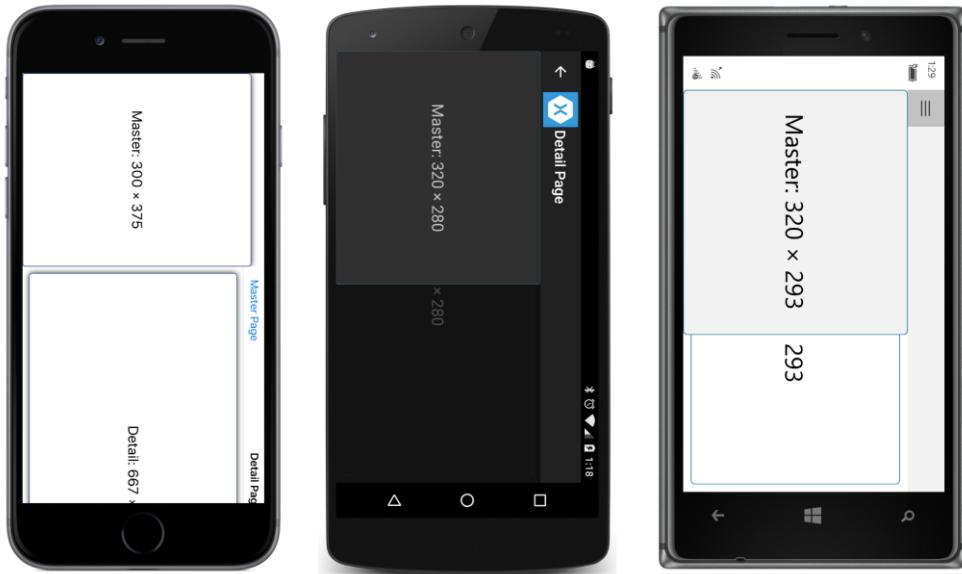
La página maestra es ahora visible. En términos de MasterBehavior enumeración, la página maestra se hace visible con una popover comportamiento, pero los tres capturas de pantalla ilustran las diferencias entre las plataformas:

- El comportamiento en iOS es una *diapositiva*. La página de detalles desliza hacia la derecha como la página maestra se desliza por la izquierda; todavía se puede ver la parte izquierda de la página de detalles.
- El Android es una *cubrir*. Es difícil de decir, porque la página de detalles se desvaneció, pero mira de cerca, y se puede ver la Marco en el detalle en el extremo derecho de la pantalla.
- Windows 10 móvil es también una *diapositiva*. Se puede ver la página de detalles detrás de la página maestra.

Por tanto iOS y Android, el ancho de la página maestra es algo menor que el ancho de la pantalla.

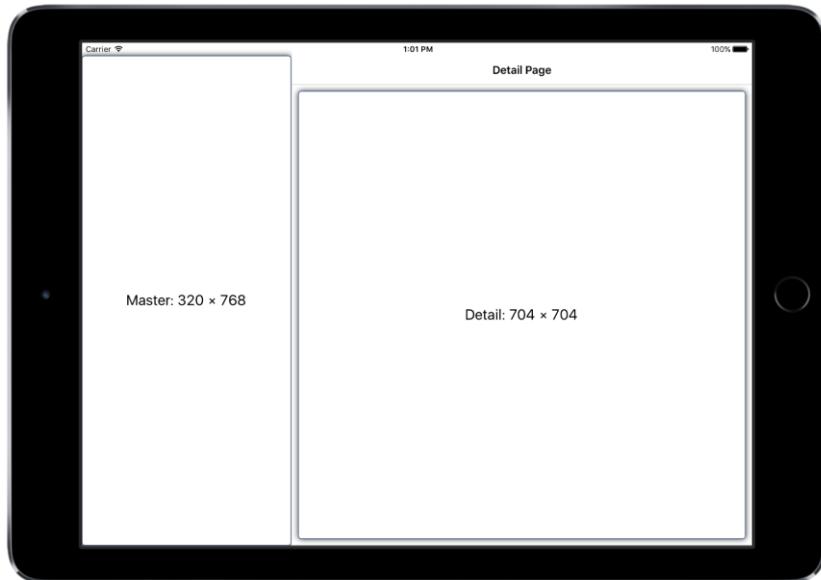
Para volver a la página de detalles en iOS, pase el dedo hacia la izquierda. En Android, deslizar la página maestra a la izquierda, toque la parte visible de la página de detalles en el extremo derecho de la pantalla, o toque en el **Espalda** triángulo en la parte inferior de la pantalla. En Windows Phone, pulse el icono del menú de nuevo o el **Espalda** flecha.

Vas a ver un comportamiento similar para las tres plataformas en modo horizontal, con la excepción de que la página principal tiene una anchura similar a la página maestra en el modo vertical, que se traduce en mucho más de la página de detalles son visibles:



Si usted experimenta con diferentes ajustes de la `MasterBehavior` propiedad de `MasterDetailPage`, Usted descubrirá que esta propiedad no tiene ningún efecto en los teléfonos. Los teléfonos tienen siempre una *popover* comportamiento. Sólo en el iPad y en las tabletas de Windows y el escritorio le verá una *división* comportamiento.

En el iPad en modo horizontal, el `MasterBehavior.Default` configuración de los resultados en una *división* comportamiento:

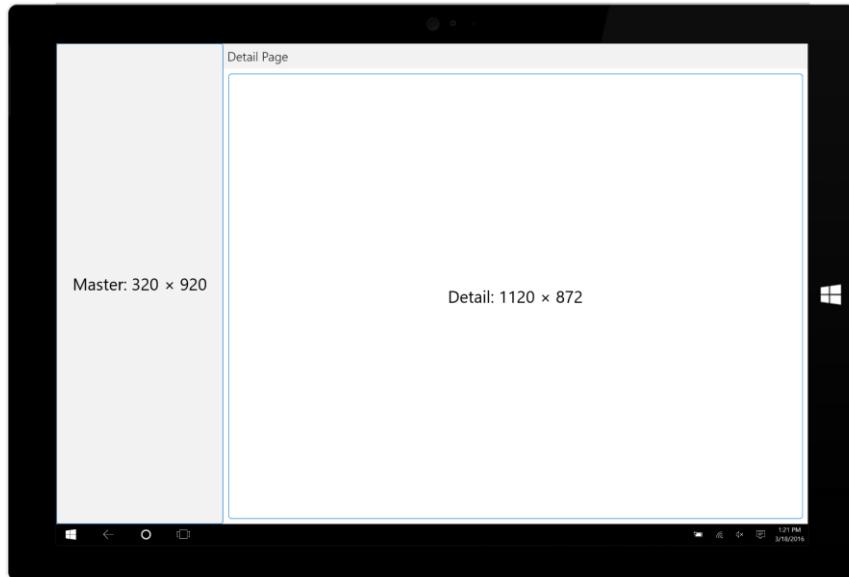


Sin embargo, se puede controlar el comportamiento. Si se establece el `MasterBehavior` propiedad a `popover`, obtendrá una página principal que se superpone a la página de detalles al igual que en el iPhone.

Para un iPad en modo vertical, el ajuste por defecto es el mismo que popover, y usted tendrá que seleccionar División o SplitOnPortrait para obtener una pantalla dividida en el modo vertical.

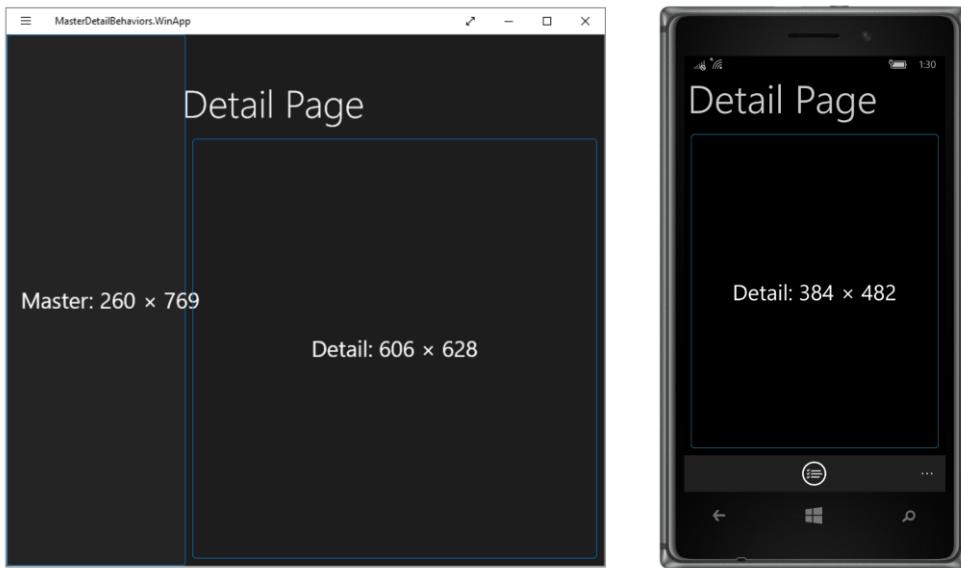
los SplitOnLandscape y SplitOnPortrait Opciones le permiten tener un comportamiento diferente para los modos vertical y horizontal. los SplitOnLandscape y SplitOnPortrait Opciones le permiten tener un comportamiento diferente para los modos vertical y horizontal. Los puntos de vista maestra y detalle comparten la pantalla en el modo horizontal, pero cuando la tableta se pone en modo vertical, la vista de detalle ocupa toda la pantalla y la página maestra lo recubre.

Aquí está el programa que se ejecuta en la superficie Pro 3 en el modo de tableta:



Este es un comportamiento de división. Usted verá un comportamiento popover si inicia el programa con la tableta en modo vertical, y se puede controlar el comportamiento con diferentes ajustes de la MasterBehavior propiedad.

La interfaz de usuario para cambiar entre el maestro y el detalle es un poco diferente en Windows 8.1 y Windows Phone 8.1. Un elemento de la barra de herramientas se entrega automáticamente al cambiar entre el maestro y el detalle:



La pantalla de Windows 8.1 muestra el comportamiento de división, pero si se establece para popover, tendrás que rightclick la pantalla para mostrar la barra de herramientas. La pantalla de Windows Phone 8.1 muestra la barra de herramientas normalmente. Usted es responsable de establecer la imagen del botón barra de herramientas y el texto asociado. La imagen y el texto son los mismos independientemente de si la vista maestra o el detalle es visible. El texto se establece a partir de la Título propiedad de la página maestra., que en este caso es "Página maestra".

El mapa de bits para el botón se ajusta desde el icono propiedad de la página maestra. (Esta icono la propiedad es en realidad definida por Página y heredados por todos los otros derivados de página.) El teléfono de Windows 8.1 proyectos 8.1 y Windows se dan tanto en una carpeta llamada Imágenes. El contenido de esta carpeta es un archivo PNG. El constructor en el archivo de código subyacente para Página principal establece que el mapa de bits a la Icono propiedad:

```

público clase parcial Página principal : Pagina de contenido
{
    público Página principal()
    {
        InitializeComponent ();

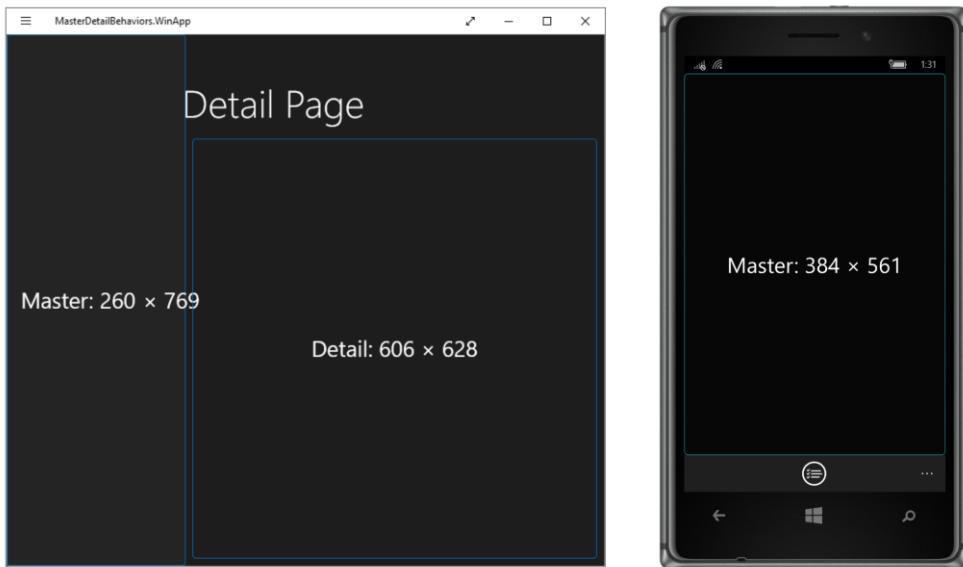
        Si ( Dispositivo .os == TargetPlatform .WinPhone ||
            Dispositivo .os == TargetPlatform .Windows)
        {
            icon = nuevo FileImageSource
            {
                file = "Imágenes / ApplicationBar.Select.png"
            };
        }
    }
}

```

Si no se establece que Icono propiedad, ya sea en el archivo de código subyacente o en el archivo XAML-la barra de herramientas

botón se mostrará en las plataformas de Windows 8.1 y Windows 8.1 teléfono sin una imagen.

Al tocar el ícono barra de herramientas cambia entre detalle y maestro:



De vuelta a la escuela

Hasta ahora, en este libro que ha visto un par de programas que utilizan una Vista de la lista para mostrar los alumnos de la Escuela de Bellas Artes. todos estos programas tienen diferentes enfoques para mostrar una visión detallada de uno de los estudiantes. los **SelectedStudentDetail** programa en el Capítulo 19, "vistas Collection", muestran la Vista de la lista en la mitad superior de la pantalla y el detalle en la mitad inferior. los **SchoolAndStu**

programa en el Capítulo 24, "navegación de la página," que se utiliza la página de navegación para visualizar el estudiante de la Vista de la lista. Ahora vamos a utilizar una **MasterDetailPage** para este trabajo y lo llaman **SchoolAndDetail**.

Una diferencia importante entre la **SchoolAndDetail** programa y **MasterDetailBehaviors** consiste en cómo se construye el programa. En lugar de tener clases separadas para la página maestra y detalle, todo lo que se consolida en una clase que deriva de **MasterDetailPage**.

Esta clase única (mostrado a continuación) se nombra **SchoolAndDetailPage**. El diseño de las páginas maestra y detalle se definen dentro de la **MasterDetailPage.Master** y **MasterDetailPage.Detail** etiquetas de propiedad de elementos.

La etiqueta raíz establece una propiedad de **MasterDetailPage** llamado **Es presentado**. Esta propiedad permite que un programa para cambiar entre vistas maestra y detalle mediante programación o de forma declarativa en XAML. El valor predeterminado es falso, lo que significa para mostrar la página de detalles, pero el elemento raíz de este archivo XAML ajustarlo en el Ciento para mostrar la página principal en el inicio:

```
< MasterDetailPage xmlns = * http://xamarin.com/schemas/2014/forms *
```

```
xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
xmlns: la escuela = "clr-espacio de nombres: SchoolOfFineArt; montaje = SchoolOfFineArt "
x: Class = "SchoolAndDetail.SchoolAndDetailPage"
Es presentado = "Cierto" >

< MasterDetailPage.Master >
  < Pagina de contenido Título = "Colegio" >
    < ContentPage.Padding >
      < OnPlatform x: TypeArguments = "Espesor" 
        iOS = "0, 20, 0, 0" />
    </ ContentPage.Padding >

    < ContentPage.Icon >
      < OnPlatform x: TypeArguments = "FileImageSource" 
        WinPhone = "Imágenes / refresh.png" />
    </ ContentPage.Icon >

    < ContentPage.BindingContext >
      < la escuela: SchoolViewModel />
    </ ContentPage.BindingContext >

    < StackLayout BindingContext = " { La unión StudentBody } " >
      < Etiqueta Texto = "(Escuelas) Encuadernación" 
        Tamaño de fuente = "Grande" 
        FontAttributes = "Negrita" 
        HorizontalTextAlignment = "Centrar" />

      < Vista de la lista x: Nombre = "vista de la lista" 
        ItemsSource = "Los estudiantes (Binding)" 
        ItemTapped = "OnListViewItemTapped" >
        < ListView.ItemTemplate >
          < DataTemplate >
            < ImageCell Fuente de imagen = " { La unión PhotoFilename } " 
              Texto = " { La unión NombreCompleto } " 
              Detalle = " {Binding GradePointAverage, 
                StringFormat = 'GPA = {0: F2}' } " />
          </ DataTemplate >
        </ ListView.ItemTemplate >
      </ Vista de la lista >
    </ StackLayout >
  </ Pagina de contenido >
</ MasterDetailPage.Master >

<! - Página detalle >
< MasterDetailPage.Detail >
  < NavigationPage >
    < x: Argumentos >
      < Pagina de contenido Título = " { FirstName } Encuadernación" 
        BindingContext = " {Binding Fuente = {x: Referencia ListView}, 
          Path = SelectedItem} " >
        < StackLayout >
          <! - Nombre >
          < StackLayout Orientación = "Horizontal" 
            HorizontalOptions = "Centrar" >
```

```

        Espaciado = "0" >
    < StackLayout.Resources >
        < ResourceDictionary >
            < Estilo Tipo de objetivo = "Etiqueta" >
                < Setter Propiedad = "Tamaño de fuente" Valor = "Grande" />
                < Setter Propiedad = "FontAttributes" Valor = "Negrita" />
            </ Estilo >
        </ ResourceDictionary >
    </ StackLayout.Resources >

        < Etiqueta Texto = "Apellido (Binding)" />
        < Etiqueta Texto = "{Binding Nombre, StringFormat = '{0}'}" />
        < Etiqueta Texto = "{Binding MiddleName, StringFormat = '{0}'}" />
    </ StackLayout >

    <!-- Foto -->
    < Imagen Fuente = "Assets/La unión PhotoFilename" 
        VerticalOptions = "FillAndExpand" />

    <!-- Sexo -->
    < Etiqueta Texto = "(Sexo Encuadernación, StringFormat = 'Sexo = {0}')"
        HorizontalOptions = "Centrar" />

    <!-- GPA -->
    < Etiqueta Texto = "{Binding GradePointAverage, StringFormat = 'GPA =
        (0: F2) '}"
        HorizontalOptions = "Centrar" />
    </ StackLayout >
    </ Página de contenido >
    </ x: Argumentos >
</ NavigationPage >
</ MasterDetailPage.Detail >
</ MasterDetailPage >

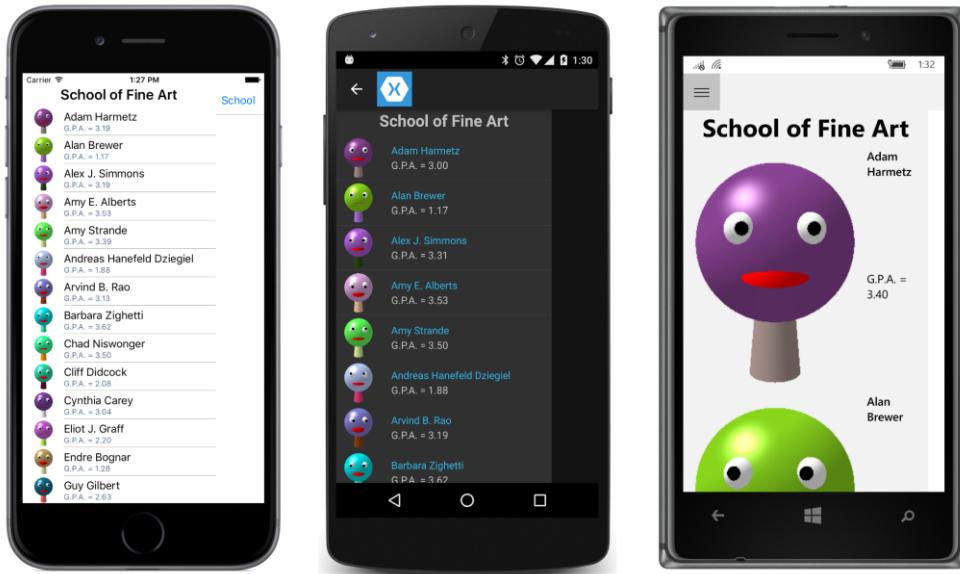
```

Observe también que el Título y Icono propiedades se establecen en la página maestra. Los proyectos de Windows 8.1 y Windows Phone 8.1 contienen una **Imágenes** directorio con una **Refrescar** ícono que también podría sugerir una operación de conmutación. La página principal también crea una instancia **SchoolViewModel** como un objeto en el Dominar-
PageBase.BindingContext etiquetas de propiedad de elementos.

Una de las ventajas de poner todo en un solo archivo XAML es que se puede establecer una unión entre las páginas maestra y **detalle de los datos**. Los **BindingContext** del **Página de contenido** que sirve como la página de detalles se une a la **Item seleccionado** propiedad de la Vista de la lista.

Aparte de esas diferencias, las definiciones de página a sí mismos son bastante similares a la **SchoolAndStudents** programa en el capítulo anterior.

El programa se inicia mostrando la página principal, que incluye la Vista de la lista con los estudiantes:



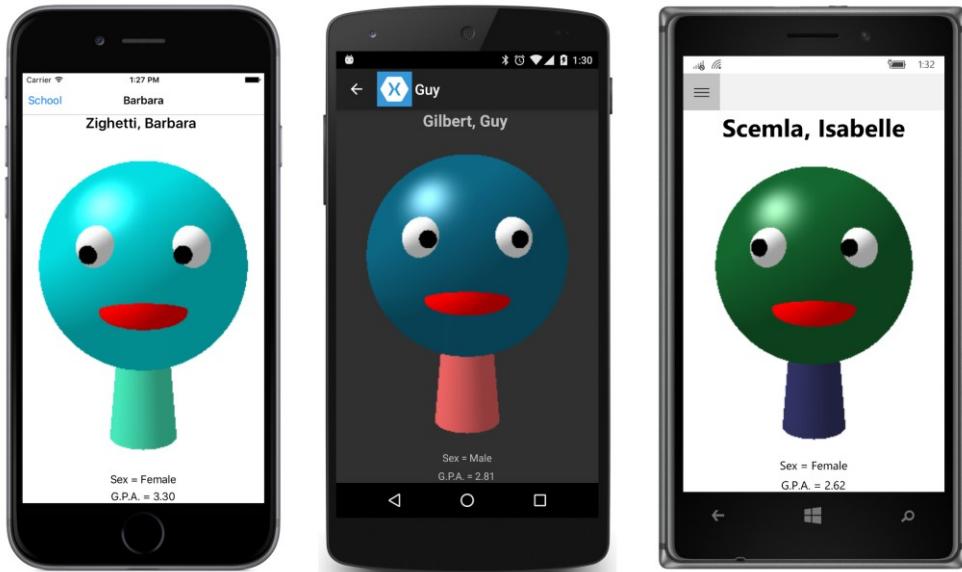
Este programa tiene otra manera de pasar de la página maestra a la página de detalles. El archivo de código subyacente contiene un controlador simple para el `ItemTapped` caso de la Vista de la lista:

```
pública clase parcial SchoolAndDetailPage : MasterDetailPage
{
    pública SchoolAndDetailPage ()
    {
        InitializeComponent ();
    }

    vacío OnListViewItemTapped ( objeto remitente, ItemTappedEventArgs args )
    {
        // Muestra la página de detalles.
        IsPresented = falso ;
    }
}
```

La diferencia entre `ItemTapped` y `ItemSelected` es que `ItemTapped` funciona incluso si el objeto ya se ha seleccionado. Los `ItemTapped` manejador de no anular la selección del material. Esto mantiene una coherencia entre el elemento seleccionado en el Vista de la lista y el contenido de la página de detalles.

Aquí está la página de detalles que se ven después de un toque:

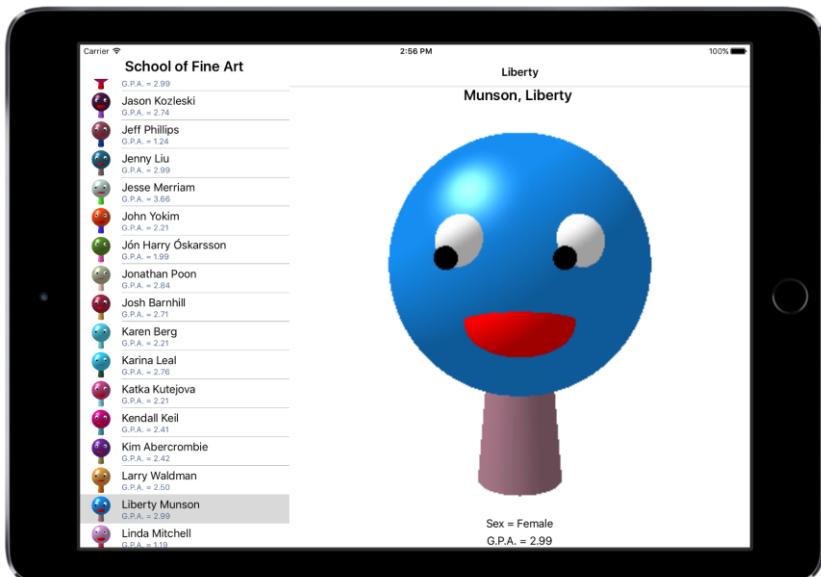


Para volver a la página maestra en IOS, deslizar derecha. En Android, desliza el dedo desde el borde izquierdo o toque la flecha en la parte superior.

En Windows 10 móvil, pulse el icono de menú en la esquina superior izquierda.

Tanto en Android y Windows Phone, tocando el Espalda flecha en la parte inferior de la pantalla se salga del programa. Ese Espalda flecha cambiará de maestro a los detalles, pero no de detalle de dominar.

Aquí está el programa que se ejecuta en el simulador del iPad de aire 2, que muestra la pantalla de lado a lado de la maestra y detalle:



Su propia interfaz de usuario

Si desea proporcionar su propia interfaz de usuario para cambiar entre las vistas maestra y detalle, es probable que también desea desactivar la interfaz proporcionada automáticamente por el MasterDetailPage. Puede hacer esto de dos maneras:

- **Selecciona el IsGestureEnabled propiedad a falso para desactivar el soporte de golpe gesto en iOS y Android.**
- **Anular el protegido ShouldShowToolbarButton método y vuelta falso para ocultar los botones de la barra en Windows 8.1 y Windows Phone 8.1.**

Sin embargo, usted no será capaz de desactivar por completo la interfaz. Ajuste de la IsGestureEnabled propiedad a falso significa que ya no se puede utilizar para cambiar golpes entre el maestro y el detalle en iOS y Android. La propiedad no afecta a los grifos, sin embargo. Tanto para iOS y Android, cuando la pantalla tiene una *popover* el comportamiento y la página maestra es la superposición de la página de detalles, se puede despedir a la página principal con un golpecito en la página de detalles de la derecha. IsGestureEnabled no desactive los grifos.

Si se establece el IsGestureEnabled propiedad a falso, tendrá que proporcionar su propia interfaz de usuario para la visualización de la vista principal de la página de detalles en iOS y Android.

El botón de la barra que acompaña a la MasterDetailPage en Windows 8.1 y Windows Phone 8.1 plataformas se adjunta a la página nativo subyacente. No se puede acceder desde la

ToolbarItems colecciones de la MasterDetailPage o de las dos páginas se establece en el Dominar y Delawarecola propiedades. anulando el ShouldShowToolbarButton y volviendo falso suprime ese botón de la barra de herramientas. Una vez más, si lo hace, debe suministrarle propia interfaz de usuario para cambiar entre vistas maestra y detalle.

Otro problema es que usted no necesita una interfaz en absoluto para cambiar entre las vistas cuando el MasterDetailPage está utilizando una *división* modo. Usted sabe que usted consigue solamente un modo de división de las tabletas iPad y Windows en tiempo de ejecución, pero si se especifica una MasterBehavior como Defecto o SplitOnLandscape. ¿cómo puede saber cuando la pantalla está en una *división* modo o *cubrir* modo?

En las tabletas de Windows en tiempo de ejecución, una llamada a la implementación base de ShouldShowToolbar-Botón Te contaré. Este método devuelve cierto para los teléfonos y las tabletas en una *cubrir* el modo, pero vuelve falso para las tabletas en una *división* modo. Sin embargo, este método sólo se ejecuta en Windows 8.1 y Windows Phone 8.1.

Para iOS, se puede determinar si el iPad está en una *cubrir* o *división* el modo por el control de las dimensiones de la página. Si la página está en el modo vertical, es cubrir; para el modo de paisaje, que es división.

Vamos a poner todo este conocimiento para su uso. Los **ColorsDetails** programa muestra todos los colores en el NamedColor recogida en una Vista de la lista en la página maestra y proporciona información detallada sobre el color seleccionado en su página de detalles. Aquí está la página de definición de maestro en primer lugar:

```
<MasterDetailPage xmlns = "http://xamarin.com/schemas/2014/forms"
    xmlns:x = "http://schemas.microsoft.com/winfx/2009/xaml"
```

```
xmlns: kit de herramientas =
    "clr-espacio de nombres: Xamarin.FormsBook.Toolkit; montaje = Xamarin.FormsBook.Toolkit "
x: Class = "ColorsDetails.ColorDetailsPage"
Es presentado = "Cíerto"
x: Nombre = "página"

< MasterDetailPage.Master >
    < Página de contenido Título = "Colores" >
        < ContentPage.Padding >
            < OnPlatform x: TypeArguments = "Espesor" 
                iOS = "0, 20, 0, 0" />
        </ ContentPage.Padding >

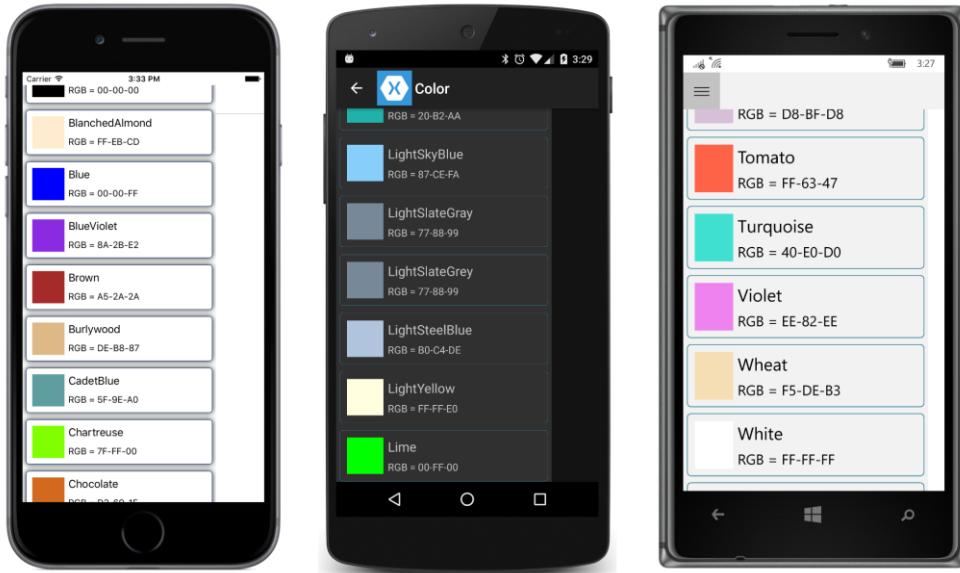
        < Vista de la lista x: Nombre = "vista de la lista" 
            SeparatorVisibility = "Ninguna" 
            ItemsSource = "{X: Estático conjunto de herramientas: NamedColor.All}" 
            ItemTapped = "OnListViewItemTapped" >
            < ListView.RowHeight >
                < OnPlatform x: TypeArguments = "x: Int32" 
                    iOS = "80" 
                    Androide = "80" 
                    WinPhone = "90" />
            </ ListView.RowHeight >

            < ListView.ItemTemplate >
                < DataTemplate >
                    < ViewCell >
                        < ContentView Relleno = "5" >
                            < Marco OutlineColor = "Acento" 
                                Relleno = "10" >
                                < StackLayout Orientación = "Horizontal" >
                                    < BoxView x: Nombre = "boxView" 
                                        Color = "{Binding Color}" 
                                        WidthRequest = "50" 
                                        HeightRequest = "50" />
                                < StackLayout >
                                    < Etiqueta Texto = "{Nombre} Encuadernación" 
                                        Tamaño de fuente = "Medio" 
                                        VerticalOptions = "StartAndExpand" />
                                    < Etiqueta Texto = "{Binding RgbDisplay, 
                                        StringFormat = 'RGB = {0}'}" 
                                        Tamaño de fuente = "Pequeña" 
                                        VerticalOptions = "CenterAndExpand" />
                                </ StackLayout >
                            </ StackLayout >
                        </ ContentView >
                    </ ViewCell >
                </ DataTemplate >
            </ ListView.ItemTemplate >
        </ Vista de la lista >
    </ Página de contenido >
</ MasterDetailPage.Master >
...
```

</ MasterDetailPage >

El margen de beneficio para este Vista de la lista es bastante similar a la de la **CustomNamedColorList** programa en el capítulo 19. En esta nueva versión, sin embargo, el **ItemTapped** caso de la Vista de la lista se maneja en el archivo de código subyacente. (Usted verá que el código en breve).

Aquí está la lista de los colores en las tres plataformas:



los Pagina de contenido que sirve como la vista de detalle tiene su **BindingContext** establecido en el Item seleccionado propiedad de la Vista de la lista. La mayor parte de los contenidos-que incluyen una BoxView el color; los valores de rojo, verde y azul; y el matiz, la saturación y luminosidad en valores están en una ScrollView. Esto es para el beneficio de los teléfonos en el modo horizontal. Los únicos elementos no en este ScrollView zona Etiqueta con el nombre del color en la parte superior de la página y una Botón En el fondo:

< MasterDetailPage ... >

```

...
< MasterDetailPage.Detail >
< NavigationPage >
< x: Arguments >
< Pagina de contenido Título = "Color" 
    BindingContext = "{Binding Fuente = {x: Referencia ListView},
    Path = SelectedItem} " >
< ContentPage.Padding >
< OnPlatform x: TypeArguments = "Espesor "
    iOS = "0, 20, 0, 0 " />
</ ContentPage.Padding >

< StackLayout >
< Etiqueta Texto = " {} La unión FriendlyName "
    Estilo = " {} DynamicResource TitleStyle "

```

```
        HorizontalTextAlignment = "Centrar" />

    < ScrollView VerticalOptions = "FillAndExpand" >
        < StackLayout >
            < BoxView Color = " {Binding Color} "
                WidthRequest = " 144 "
                HeightRequest = " 144 "
                VerticalOptions = " CenterAndExpand "
                HorizontalOptions = " Centrar " />

            < StackLayout VerticalOptions = " CenterAndExpand "
                HorizontalOptions = " Centrar " >
                < StackLayout.Resources >
                    < ResourceDictionary >
                        < Estilo Tipo de objetivo = " Etiqueta " >
                            < Setter Propiedad = " HorizontalTextAlignment "
                                Valor = " Fin " />
                        </ Estilo >
                    </ ResourceDictionary >
                </ StackLayout.Resources >

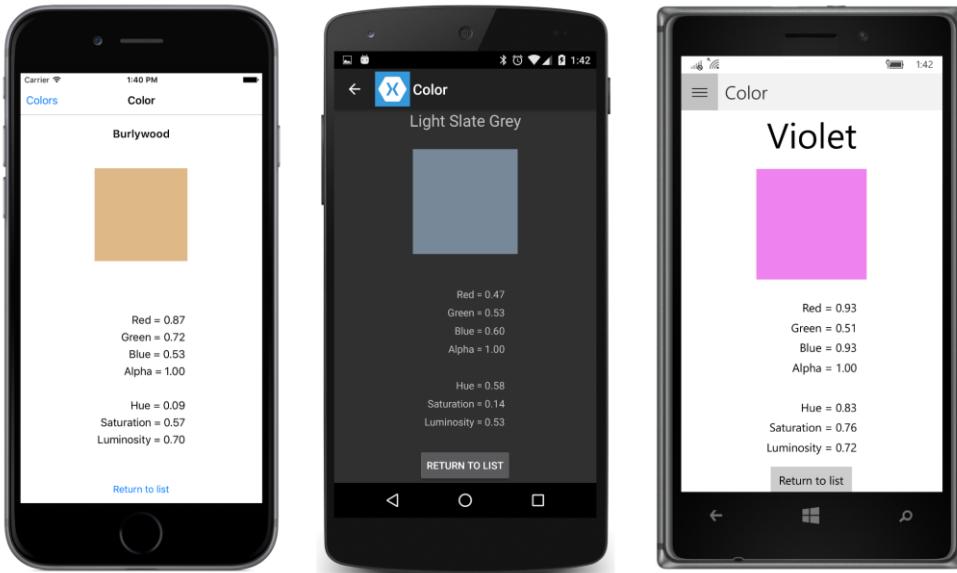
                < Etiqueta Texto = " {Binding Color.R,
                    StringFormat = 'Red = {0: F2}'} " />
                < Etiqueta Texto = " {Binding Color.G,
                    StringFormat = 'Verde = {0: F2}'} " />
                < Etiqueta Texto = " {Binding Color.B,
                    StringFormat = 'Azul = {0: F2}'} " />
                < Etiqueta Texto = " {Binding Color.A,
                    StringFormat = 'Alpha = {0: F2}'} " />
                < Etiqueta Texto = " " />
                < Etiqueta Texto = " {Binding Color.Hue,
                    StringFormat = 'Hue = {0: F2}'} " />
                < Etiqueta Texto = " {Binding Color.Saturation,
                    StringFormat = 'Saturación = {0: F2}'} " />
                < Etiqueta Texto = " {Binding Color.Luminosity,
                    StringFormat = 'Luminosidad = {0: F2}'} " />
            </ StackLayout >
        </ StackLayout >
    </ ScrollView >

    < Botón x: Nombre = " returnButton "
        Texto = " Volver a la lista "
        HorizontalOptions = " Centrar "
        hecho clic = " OnReturnButtonClicked " >
        < Button.IsEnabled >
            < Unión Fuente = " {X: Página de referencia} "
                Camino = " Es presentado " >
                < Binding.Converter >
                    < kit de herramientas: BooleanNegationConverter />
                </ Binding.Converter >
            </ Unión >
        </ Button.IsEnabled >
```

```
</ Botón >
</ StackLayout >
</ Página de contenido >
</ x: Argumentos >
</ NavigationPage >
</ MasterDetailPage.Detail >
</ MasterDetailPage >
```

los Botón en la parte inferior tiene una hecho clic controlador de eventos en el archivo de código subyacente, por supuesto, pero también notan los datos de unión a su Está habilitado propiedad. La fuente de los enlace de datos es la IsPresentedChanged propiedad de la MasterDetailPage. Si Es presentado es cierto -que significa que vista maestra se visualiza entonces la Botón está desactivado. (Si desea hacer algo similar en el código, MasterDetailPage define una IsPresentedChanged evento.)

Puedes ver el Botón en la parte inferior de la vista de detalle para volver a la vista maestra:



El archivo de código subyacente se ocupa de los controladores de eventos para el Vista de la lista y botón (hacia el final del archivo). Estos limitan a establecer la Es presentado propiedad a falso y cierto, respectivamente, y no tienen efecto cuando el MasterDetailPage es en división modo:

```
público clase parcial ColorDetailsPage : MasterDetailPage
{
    público ColorDetailsPage ()
    {
        InitializeComponent ();

        IsGestureEnabled = falso ;

        // Procesamiento especial para iPads.
        Si ( Dispositivo.os == TargetPlatform.iOS &&
```

```
Dispositivo.Idiom == TargetIdiom.Tableta)
{
    SizeChanged += (remitente, args) =>
    {
        Botón de modo vertical // Habilitar.
        returnButton.Visible = Altura > Anchura;
    };
}

público bool anulación ShouldShowToolbarButton ()
{
    // Sólo funciona para plataformas Windows y Windows Phone.
    returnButton.Visible = base.ShouldShowToolbarButton();

    falso retorno ;
}

vacío OnListViewItemTapped ( objeto remitente, ItemTappedEventArgs args )
{
    IsPresented = falso ;
}

vacío OnReturnButtonClicked ( objeto remitente, EventArgs args )
{
    IsPresented = cierto ;
}
}
```

Las partes más interesantes del archivo de código subyacente están en el constructor y la anulación de la ShouldShowToolbarButton. Estas secciones de código intento de dos puestos de trabajo:

En primer lugar, deshabilitar la interfaz de usuario existente para la conmutación entre las vistas maestra y detalle mediante el establecimiento IsGestureEnabled a falso y volviendo falso de ShouldShowToolbarButton. Esto significa que no hay ningún elemento barra de herramientas se muestra en el Windows 8.1 y Windows Phone 8.1 plataformas. los

MasterDetailPage aún requiere que una Título fijarse en el Página de contenido que sirve como la vista principal, pero que Título no se utiliza en cualquier parte de estas plataformas.

El segundo trabajo es ocultar que Botón por completo cuando la MasterDetailPage está a la vista dividida. los SizeChanged manejador de la página se encuentra en el constructor cuando el programa se está ejecutando en un iPad, y se establece el Es visible propiedad a cierto sólo si las dimensiones de la página indican el modo de retrato. los ShouldShowToolbarButton anulación maneja tabletas de Windows mostrando el Botón Si la implementación base de ShouldShowToolbarButton devoluciones cierto.

Esa es una manera de poner en práctica su propia interfaz de usuario para cambiar entre vistas maestra y detalle. los MasterDetailTaps programa muestra otro enfoque. Este programa es similar a la MasterDetailBehavior programa que se inició este capítulo, pero con las definiciones de los puntos de vista maestra y detalle combinados en un solo archivo XAML. Este nuevo programa desactiva la interfaz de usuario existente para la transición entre las vistas maestra y detalle y lo reemplaza con simples toques.

MasterDetailTapsPage deriva de **MasterDetailPage** e incluye similares Marco y Etiqueta elementos que el programa anterior:

```
< MasterDetailPage xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns:x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " MasterDetailTaps.MasterDetailTapsPage "
    Título = " Página de demostración " >

    < MasterDetailPage.Master >
        < Pagina de contenido Título = " Dominar "
            Relleno = " 10 "
            x: Nombre = " página principal " >

            < Marco OutlineColor = " Acento "
                Color de fondo = " Transparente " >
                < Frame.GestureRecognizers >
                    < TapGestureRecognizer aprovechado = " OnMasterTapped " />
                </ Frame.GestureRecognizers >

                < StackLayout Orientación = " Horizontal "
                    Espaciado = " 0 "
                    HorizontalOptions = " Centrar "
                    VerticalOptions = " Centrar " >

                    < Etiqueta Texto = " {Binding Fuente = {x: masterpage Referencia},
                        Path = Ancho,
                        StringFormat = "Maestro: {0: F0}" }
                        Tamaño de fuente = " Grande " />

                    < Etiqueta Texto = " {Binding Fuente = {x: masterpage Referencia},
                        Path = Altura,
                        StringFormat = "& # X00D7; {0: F0} "
                        Tamaño de fuente = " Grande " />
                </ StackLayout >
            </ Marco >
        </ Pagina de contenido >
    </ MasterDetailPage.Master >

    < MasterDetailPage.Detail >
        < NavigationPage >
            < x: Argumentos >
                < Pagina de contenido Título = " Detalle "
                    Relleno = " 10 "
                    x: Nombre = " detalle " >
                    < ContentPage.Padding >
                        < OnPlatform x: TypeArguments = " Espesor "
                            iOS = " 0, 20, 0, 0 " />
                    </ ContentPage.Padding >

                    < Marco OutlineColor = " Acento "
                        Color de fondo = " Transparente " >
                        < Frame.GestureRecognizers >
                            < TapGestureRecognizer aprovechado = " OnDetailTapped " />
                        </ Frame.GestureRecognizers >
                </ Pagina de contenido >
            </ x: Argumentos >
        </ NavigationPage >
    </ MasterDetailPage.Detail >
```

```

< StackLayout Orientación = " Horizontal "
    Espaciado = " 0 "
    VerticalOptions = " CenterAndExpand "
    HorizontalOptions = " Centrar " >
    < Etiqueta Texto = " {Binding Fuente = {x: detalle de referencia}},
        Path = Ancho,
        StringFormat = 'Detalle: {0: F0}' >
        Tamaño de fuente = " Grande " />

    < Etiqueta Texto = " {Binding Fuente = {x: detalle de referencia}},
        Path = Altura,
        StringFormat = '& # X00D7; {0: F0}' >
        Tamaño de fuente = " Grande " />
    </ StackLayout >
</ Marco >
</ Página de contenido >
</ x: Argumentos >
</ NavigationPage >
</ MasterDetailPage.Detail >
</ MasterDetailPage >

```

Observe la TapGestureRecognizer adjunto a Marco elemento en ambos las páginas maestra y detalle.

Observe también que el Color de fondo de cada Marco se establece en Transparente. Esto es para el beneficio de las plataformas de Windows. El fondo por defecto de un Marco en estas plataformas es nulo, que permite a los grifos caen a través del elemento subyacente. Ajuste del fondo de Transparente no cambia la apariencia, pero captura los grifos.

los aprovechado manipuladores establecen simplemente Es presentado:

```

pública clase parcial MasterDetailTapsPage : MasterDetailPage
{
    pública MasterDetailTapsPage ()
    {
        InitializeComponent ();

        // interfaz swipe Desactivar.
        IsGestureEnabled = falso ;
    }

    pública bool anulación ShouldShowToolbarButton ()
    {
        // Ocultar barra de herramientas en plataformas Windows.
        falso retorno ;
    }

    vacío OnMasterTapped ( objeto remitente, EventArgs args )
    {
        // excepciones de captura cuando se ajuste IsPresented en modo dividido.
        tratar
    }
}

```

```
        IsPresented = false ;
    }
    captura
    {
    }
}

void OnDetailTapped ( objeto remitente, EventArgs args)
{
    IsPresented = cierto ;
}
}
```

La interfaz de usuario normal se desactiva al igual que en el programa anterior, pero no se requiere ninguna lógica para ocultar la nueva interfaz de usuario en *división* modo.

los tratar y captura bloque en el OnMasterTapped método se utiliza para evitar una InvalidOperationException que se produce en Windows y iPads en modo dividido. El mensaje de error que acompaña la excepción indica “No se puede cambiar la hora de establecer IsPresented Split.”

TabPage

TabPage se deriva de la clase abstracta MultiPage < página >. Se mantiene una colección de niños de tipo Página. Sólo uno de los cuales es completamente visible a la vez. TabbedPane identifica cada niño por una serie de pestanas en la parte superior o inferior de la página. Una aplicación de iOS que utiliza una TabbedPage debe incluir un ícono para cada ficha; de lo contrario, Apple no va a aceptar el programa de la App Store. Este ícono se establece a través de cada página Icono propiedad.

MultiPage < T > define todas las importantes propiedades y eventos para TabbedPage, el más importante de los cuales es:

- Niños propiedad de tipo IList < T >.

Normalmente, esto se llena Niños colección con objetos de la página.

Sin embargo, puede utilizar TabbedPage de una manera un tanto diferente al observar que MultiPage < T > es bastante similar a ItemsView < T >, la clase base de Vista de la lista, en que define:

- el ItemsSource propiedad de tipo IEnumerable, y
- el ItemTemplate propiedad de tipo DataTemplate.

Si proporciona una IEnumerable colección de objetos con propiedades públicas adecuadas para enlaces de datos y una plantilla con un tipo de página como elemento raíz, entonces los niños se generan de forma dinámica. Los BindingContext de cada página generada se ajusta igual a la objeto particular de ItemsSource.

MultiPage < T > define dos propiedades que pueden ayudar a su aplicación perder de vista que la página en el Niños colección el usuario está viendo actualmente:

- Página actual de tipo T (Página para TabbedPage).
- Item seleccionado de tipo objeto, en referencia a un objeto en el ItemsSource colección.

Ambas propiedades son gettable y ajustable.

Multipage <T> también define dos eventos:

- PagesChanged se activa cuando el ItemsSource cambios de recolección
- CurrentPageChanged se activa cuando la página visitada.

Por lo general, va a añadir Pagina de contenido derivados directamente a la Niños colección. Si desea utilizar TabbedPage para mostrar una colección de páginas similares a base de un conjunto de datos, se puede establecer como alternativa al ItemsSource propiedad para que la recogida y definir una página mediante el uso de ItemTemplate, pero este enfoque debe ser evitado en IOS.

páginas de fichas discretos

El uso más común de TabbedPage es para navegar entre las diferentes funciones dentro de una aplicación, lo que normalmente significa cada pestaña presenta un tipo diferente de página. Es común que estas páginas sean relacionados de alguna manera, tal vez las páginas múltiples para los ajustes de equilibrio de aplicación si no tienen el mismo aspecto.

los DiscreteTabbedColors programa tiene tres pestañas: la primera muestra una lista de los colores Xamarin.Forms incorporadas, el segundo muestra una lista de los colores de la NamedColor clase en el Xamarin.FormsBook.Toolkit (introdujo en capítulos anteriores), y el tercero contiene un color-probador (con el que se puede seleccionar valores RGB arbitrarias de vista previa).

los DiscreteTabbedColors programa comienza con tres Pagina de contenido derivados. La primera es codeonly y consiste en una simple lista de los colores Xamarin.Forms estándar.

```
clase BuiltinColorsPage : Pagina de contenido
{
    público BuiltinColorsPage ()
    {
        title = "Incorporado";
        icon = Dispositivo .OnPlatform ("ic_action_computer.png", nulo , nulo );
        Relleno = nuevo Espesor (5, Dispositivo .OnPlatform (20, 5, 5), 5, 5);
        doble fontSize = Dispositivo .GetNamedSize (NamedSize .Grande, tipo de (Etiqueta));
        content = nuevo ScrollView
        {
            content = nuevo StackLayout
            {
                Espaciado = 0,
                Los niños =
                {
                    nuevo Etiqueta
                    {
                        text = "Blanco",
                        TextColor = Color .Blanco,
                        FontSize = fontSize
                    }
                }
            }
        }
    }
}
```

```
        },
        ...
    },
    nuevo Etiqueta
    {
        text = "Púrpura" ,
        TextColor = Color .Púrpura,
        FontSize = fontSize
    }
}
}
}
}
```

Observe que el Título propiedad se establece. Esto es esencial para el texto de pestaña en todas las plataformas. El código también establece el Icono propiedad para iOS. El icono en particular es parte del conjunto de iconos de Android que se describen en el capítulo 13, "Los mapas de bits", y es de 32 píxeles cuadrados.

Título propiedad y Icono propiedad para iOS:

```
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    xmlns: kit de herramientas =
        "clr-espacio de nombres: Xamarin.FormsBook.Toolkit; montaje = Xamarin.FormsBook.Toolkit"
    x: Class = " DiscreteTabbedColors.NamedColorsPage "
    Título = " Toolkit "
< ContentPage.Icon >
    < OnPlatform x: TypeArguments = " FileImageSource "
        iOS = " ic_action_storage.png "
    </ OnPlatform >
</ ContentPage.Icon >

< Vista de la lista ItemsSource = " {x: Estático conjunto de herramientas: NamedColor.All} " >
    < ListView.RowHeight >
        < OnPlatform x: TypeArguments = " x: Int32 "
            iOS = " 80 "
            Androide = " 80 "
            WinPhone = " 90 "
        </ OnPlatform >
    </ ListView.RowHeight >

    < ListView.ItemTemplate >
        < DataTemplate >
            < ViewCell >
                < ContentView Relleno = " 5 " >
                    < StackLayout Orientación = " Horizontal " >
                        < BoxView x: Nombre = " boxView "
                            Color = " {Binding Color} "
                            WidthRequest = " 50 "
                            HeightRequest = " 50 "
                        </ BoxView >
                    < StackLayout >
                        < Etiqueta Texto = " {Nombre} Encuadernación " >
                            Tamaño de fuente = " Medio "
                        </ Etiqueta >
                    </ StackLayout >
                </ ContentView >
            </ ViewCell >
        </ DataTemplate >
    </ ListView.ItemTemplate >
</ Vista de la lista >
```

```

        VerticalOptions = " StartAndExpand " />
    < Etiqueta Texto = "{Binding RgbDisplay, StringFormat = 'RGB = {0}'} "
        Tamaño de fuente = " Pequeña "
        VerticalOptions = " CenterAndExpand " />
    </ StackLayout >
    </ StackLayout >
</ ContentView >
</ ViewCell >
</ DataTemplate >
</ ListView.ItemTemplate >
</ Vista de la lista >
</ Página de contenido >

```

La tercera página contiene un trío de deslizador elementos para seleccionar un color, como se ha visto antes:

```

< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    xmlns: kit de herramientas =
        "clr-espacio de nombres: Xamarin.FormsBook.Toolkit; montaje = Xamarin.FormsBook.Toolkit"
    x: Class = " DiscreteTabbedColors.ColorTestPage "
    Título = " Prueba " >

< ContentPage.Icon >
    < OnPlatform x: TypeArguments = " FileImageSource "
        iOS = " ic_action_gamepad.png " />
</ ContentPage.Icon >

< StackLayout Relleno = " 20, 40 " >
    < StackLayout.BindingContext >
        < kit de herramientas: ColorViewModel Color = " gris " />
    </ StackLayout.BindingContext >

    < Etiqueta Texto = "{Binding Red, StringFormat = 'Rojo = {0: F2}'} "
        HorizontalOptions = " Centrar " />

    < deslizador Valor = " 0 La unión Rojo " />

    < Etiqueta Texto = "{Binding Green, StringFormat = 'Verde = {0: F2}'} "
        HorizontalOptions = " Centrar " />

    < deslizador Valor = " {Verde} Encuadernación " />

    < Etiqueta Texto = "{Binding azul, StringFormat = 'Azul = {0: F2}'} "
        HorizontalOptions = " Centrar " />

    < deslizador Valor = " 0 La unión Azul " />

    < BoxView Color = "{Binding Color} "
        VerticalOptions = " FillAndExpand " />
</ StackLayout >
</ Página de contenido >

```

Aquí está la DiscreteTabbedColorsPage. Note el elemento raíz del TabbedPage. Este archivo XAML no hace sino aumentar los casos de estos tres tipos de página a la Niños colección de la TabbedPage:

```
< TabbedPage xmlns = " http://xamarin.com/schemas/2014/forms "
```

```

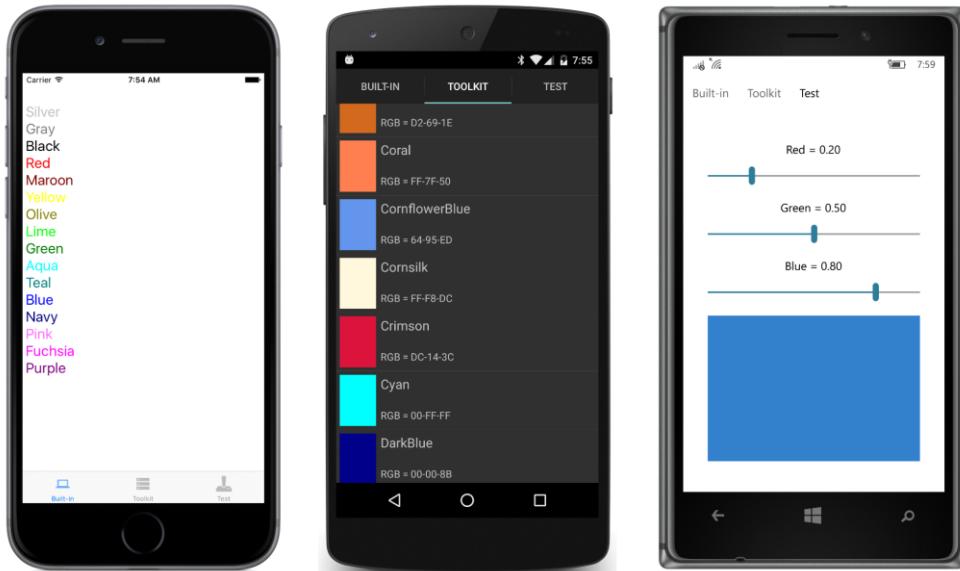
<xmlns: x = "http://schemas.microsoft.com/winfx/2009/xaml"
  xmlns: locales = "CLR-espacio de nombres: DiscreteTabbedColors"
  x: Class = "DiscreteTabbedColors.DiscreteTabbedColorsPage">

<locales: BuiltInColorsPage />
<locales: NamedColorsPage />
<locales: ColorTestPage />

</TabbedPage>

```

Aquí están las tres lengüetas de las tres plataformas:



En iOS, las pestañas están en la parte inferior, identificado con texto e iconos, y la ficha seleccionada se resalta. Tanto en Android y Windows Mobile 10 se muestran las pestañas en la parte superior de la pantalla, pero destacan la ficha seleccionada de diferentes maneras.

los StudentNotes programa cuenta con una página de inicio que muestra todos los estudiantes en una Vista de la lista, pero seleccionando un estudiante de esta lista hace que el programa para navegar a una TabbedPage. La página tiene tres pestañas: la primera muestra información textual sobre el estudiante, el segundo muestra la fotografía del alumno, y el tercero muestra una Editor que permite a un maestro u otro administrador de la escuela para entrar en algunas notas sobre el estudiante. (Esta característica hace uso de la notas propiedad en el Estudiante clase en el SchoolOffineArt biblioteca.)

los Aplicación clase en el StudentNotes pasa programa de la propiedades diccionario definido por aplicación al SchoolViewModel constructor, y también pasa por el propiedades diccionario para la SaveNotes método de la ViewModel cuando el programa se va a dormir, posiblemente en preparación para ser terminado:

```
público clase Aplicación : Solicitud
```

```
{  
    público App ()  
    {  
        ViewModel = nuevo SchoolViewModel (Propiedades);  
  
        MainPage = nuevo NavigationPage ( nuevo StudentNotesHomePage ());  
    }  
  
    público SchoolViewModel ViewModel  
    {  
        conjunto privado ; obtener ;  
    }  
  
    protegido override void OnStart ()  
    {  
        // manipulador cuando se inicia el app  
    }  
  
    protegido override void OnSleep ()  
    {  
        ViewModel.SaveNotes (Properties);  
    }  
  
    protegido override void En resumen()  
    {  
        // manejar cuando sus hojas de vida de aplicaciones  
    }  
}
```

La página de inicio debe resultar familiar a estas alturas. Simplemente muestra todos los estudiantes en una Vista de la lista:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "  
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "  
    xmlns: locales = "clr-espacio de nombres: StudentNotes; montaje = StudentNotes "  
    x: Class = " StudentNotes.StudentNotesHomePage "  
    Título = "estudiantes "  
    BindingContext = "{Binding Fuente = {x: Estático Application.Current},  
    Path = ViewModel} " >  
  
< StackLayout BindingContext = " () La unión StudentBody " >  
    < Etiqueta Texto = " (Escuela) Encuadernación "  
        Tamaño de fuente = " Grande "  
        FontAttributes = " Negrita "  
        HorizontalTextAlignment = " Centrar " />  
  
    < Vista de la lista x: Nombre = " vista de la lista "  
        ItemsSource = " Los estudiantes (Binding) "  
        itemSelected = " OnListViewItemSelected " >  
        < ListView.ItemTemplate >  
            < DataTemplate >  
                < ImageCell Fuente de imagen = " () La unión PhotoFilename "  
                    Texto = " () La unión NombreCompleto "  
                    Detalle = " {Binding GradePointAverage,  
                    StringFormat = 'GPA = {0: F2}'} " />  
            </ DataTemplate >  
        </ ListView.ItemTemplate >
```

```

</ ListView.ItemTemplate >
</ Vista de la lista >
</ StackLayout >
</ Página de contenido >

```

El archivo de código subyacente contiene el itemSelected controlador para el Vista de la lista para desplazarse hasta StudentNotesDataPage, el establecimiento de la página BindingContext al seleccionado Estudiante objeto:

```

pública clase parcial StudentNotesHomePage : Página de contenido
{
    pública StudentNotesHomePage ()
    {
        InitializeComponent ();
    }

    vacío asíncrono On.listViewItemSelected ( objeto remitente, SelectedItemChangedEventArgs args )
    {
        Si (Args.SelectedItem = nulo )
        {
            listView.SelectedItem = nulo ;

            esperar Navigation.PushAsync ( nuevo StudentNotesDataPage
            {
                BindingContext = args.SelectedItem
            });
        }
    }
}

```

los StudentNotesDataPage deriva de TabbedPage. Dentro de las etiquetas de inicio y final de la TabbedPage, Tres Página de contenido definiciones se añaden a la Niños propiedad de TabbedPage. Cada uno tiene su Titulo propiedad establecida en el texto a utilizar en la pestaña, y Icono definiciones se incluyen para iOS:

```

< TabbedPage xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " StudentNotes.StudentNotesDataPage "
    Titulo = " datos del estudiante " >

    < Página de contenido Titulo = " información " >
        < ContentPage.Icon >
            < OnPlatform x: TypeArguments = " FileImageSource "
                iOS = " ic_action_about.png " />
        </ ContentPage.Icon >

        < StackLayout >
            < Etiqueta Texto = " {0} La unión NombreCompleto "
                Tamaño de fuente = " Grande "
                HorizontalOptions = " Centrar " />

            < StackLayout Espaciado = " 12 "
                VerticalOptions = " CenterAndExpand "
                HorizontalOptions = " Centrar " >
                < StackLayout.Resources >
                    < ResourceDictionary >

```

```
< Estilo Tipo de objetivo = "Etiqueta" >
    < Setter Propiedad = "Tamaño de fuente" Valor = "Grande" />
</ Estilo >
</ ResourceDictionary >
</ StackLayout.Resources >

< Etiqueta Texto = "{Binding Apellidos,
StringFormat = 'Apellido: {0}'}" />

< Etiqueta Texto = "{Binding Nombre,
StringFormat = 'Nombre: {0}'}" />

< Etiqueta Texto = "{Binding MiddleName,
StringFormat = 'Nombre del Medio: {0}'}" />

< Etiqueta Texto = "{Sexo Encuadernación,
StringFormat = 'Sexo: {0}'}" />

< Etiqueta Texto = "{Binding GradePointAverage,
StringFormat = 'GPA = {0: F2}'}" />
</ StackLayout >
</ StackLayout >
</ Pagina de contenido >

< Pagina de contenido Título = "Foto" >
    < ContentPage.Icon >
        < OnPlatform x: TypeArguments = "FileImageSource"
            iOS = "ic_action_person.png" />
    </ ContentPage.Icon >

    < StackLayout >
        < Etiqueta Texto = "{} La unión NombreCompleto"
            Tamaño de fuente = "Grande"
            HorizontalOptions = "Centrar" />

        < Imagen Fuente = "{} La unión PhotoFilename"
            VerticalOptions = "FillAndExpand" />
    </ StackLayout >
</ Pagina de contenido >

< Pagina de contenido Título = "notas" >
    < ContentPage.Icon >
        < OnPlatform x: TypeArguments = "FileImageSource"
            iOS = "ic_action_edit.png" />
    </ ContentPage.Icon >

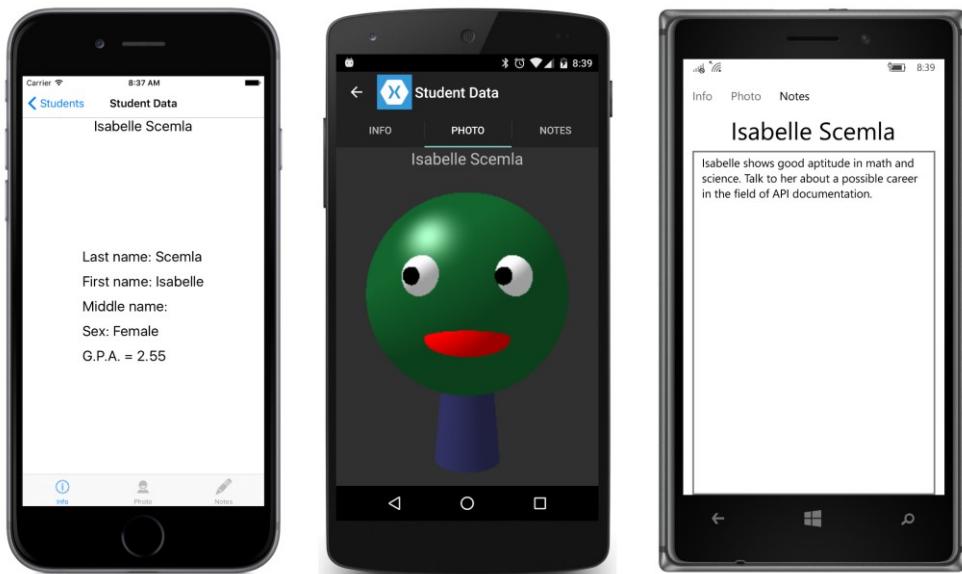
    < StackLayout >
        < Etiqueta Texto = "{} La unión NombreCompleto"
            Tamaño de fuente = "Grande"
            HorizontalOptions = "Centrar" />

        < Editor Texto = "{} Notas Encuadernación"
            Tectedo = "Texto"
            VerticalOptions = "FillAndExpand" />
    </ StackLayout >
```

```
</ StackLayout >
</ Página de contenido >
</ TabbedPage >
```

Esto tal vez no sea suficiente información para repartidos en tres páginas, pero se puede imaginar fácilmente situaciones en las que este enfoque sería ideal.

He aquí cómo las tres pestañas se ven en las tres plataformas:



Puede navegar de nuevo a la lista de los estudiantes de la forma habitual: Al hacer uso de la flecha hacia la izquierda en la parte superior de la pantalla en iOS y Android, o pulsando el **Espalda flecha** en la parte inferior de la pantalla en Android y Windows Mobile 10.

El uso de un ItemTemplate

los TabbedPage también se puede utilizar para presentar un pequeño conjunto de datos, cada elemento de los cuales es una página separada identificado por una pestaña. Esto se hace mediante el establecimiento de la `ItemsSource` propiedad de `TabbedPage` y la especificación de una `ItemTemplate` para la prestación de cada página.

los **MultiTabbedColors** proyecto contiene una sola clase de página que se ha añadido al proyecto como una `Página de contenido`, pero que después se modifica para ser un `TabbedPage`. El proyecto también cuenta con una referencia a la `Xamarin.FormsBook.Toolkit` biblioteca.

Observe que el elemento raíz del archivo XAML establece el `ItemsSource` propiedad de `TabbedPage` a la colección disponible en la `NamedColor.All` propiedad estática. El resto del archivo define la `ItemTemplate` propiedad. Los `TabbedPage.ItemTemplate` etiquetas de propiedad de elementos encierran un par de `DataTemplate` etiquetas, en las que aparece una definición de página, a partir de `Página de contenido`. Los datos

enlaces de referencia propiedades de los objetos en el ItemsSource colección, en este caso las propiedades de NamedColor:

```
<TabbedPage xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    xmlns: kit de herramientas =
        "clr-espacio de nombres: Xamarin.FormsBook.Toolkit; montaje = Xamarin.FormsBook.Toolkit"
    x: Class = "MultiTabbedColors.MultiTabbedColorsPage"
    ItemsSource = "{X: Estático conjunto de herramientas: NamedColor.All}" >

    < TabbedPage.ItemTemplate >
        < DataTemplate >
            < Pagina de contenido Título = "{Nombre} Encuadernación" >
                < ContentPage.Padding >
                    < OnPlatform x: TypeArguments = " Espesor "
                        iOS = "0, 20, 0, 0" />
                </ ContentPage.Padding >

                < StackLayout >
                    < Etiqueta Texto = "{0} La unión FriendlyName" 
                        Estilo = "{} DynamicResource TitleStyle"
                        HorizontalTextAlignment = "Centrar" />

                    < ScrollView VerticalOptions = " FillAndExpand" >
                        < StackLayout >
                            < BoxView Color = "{Binding Color}"
                                WidthRequest = "144"
                                HeightRequest = "144"
                                VerticalOptions = " CenterAndExpand"
                                HorizontalOptions = " Centrar" />

                            < StackLayout VerticalOptions = " CenterAndExpand"
                                HorizontalOptions = " Centrar" >
                                < StackLayout.Resources >
                                    < ResourceDictionary >
                                        < Estilo Tipo de objetivo = " Etiqueta" >
                                            < Setter Propiedad = " HorizontalTextAlignment "
                                                Valor = " Fin" />
                                        </ Estilo >
                                    </ ResourceDictionary >
                                </ StackLayout.Resources >

                                < Etiqueta Texto = "{Binding Color.R,
                                    StringFormat = 'Red = {0: F2}'}" />
                                < Etiqueta Texto = "{Binding Color.G,
                                    StringFormat = 'Verde = {0: F2}'}" />
                                < Etiqueta Texto = "{Binding Color.B,
                                    StringFormat = 'Azul = {0: F2}'}" />
                                < Etiqueta Texto = "{Binding Color.A,
                                    StringFormat = 'Alpha = {0: F2}'}" />
                                < Etiqueta Texto = "" />
                                < Etiqueta Texto = "{Binding Color.Hue,
                                    StringFormat = 'Hue = {0: F2}'}" />
                                < Etiqueta Texto = "{Binding Color.Saturation,
                                    StringFormat = 'Saturation = {0: F2}'}" />
                            </ StackLayout >
                        </ ScrollView >
                    </ ContentPage >
                </ TabbedPage >
```

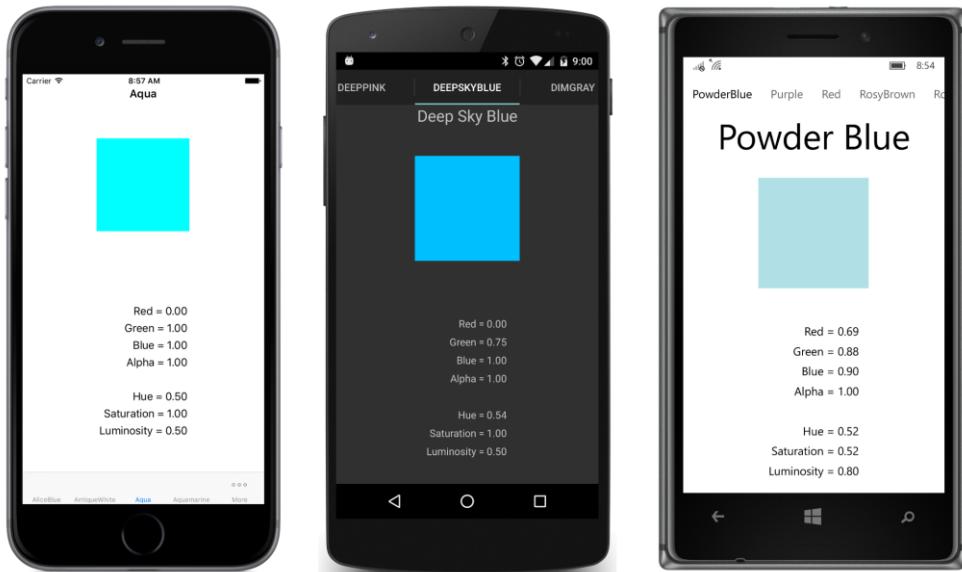
```

        StringFormat = "Saturación = {0: F2}" />
    < Etiqueta Texto = " {Binding Color.Luminosity,
        StringFormat = "Luminosidad = {0: F2}" />
    </ StackLayout >
</ StackLayout >
</ ScrollView >
</ StackLayout >
</ Pagina de contenido >
</ DataTemplate >
</ TabbedPage.ItemTemplate >
</ TabbedPage >
```

Para evitar sobreescribir la barra de estado en la parte superior de la pantalla iOS, configurar Relleno sobre el Pagina de contenido plantilla en lugar de en el TabbedPage sí mismo.

Selecciona el Título propiedad de este Pagina de contenido plantilla para el texto que desea que aparezca en las pestañas para identificar cada página. Observe que el Título está unido a la Nombre propiedad de NamedColor, pero el contenido de la página también incluyen una Etiqueta con un TextStyle para mostrar la Nombre amigable propiedad, que es similar a la Nombre propiedad, pero incluye espacios, si el nombre del color se compone de varias palabras.

Aquí está **TabbedColors** que se ejecuta en las tres plataformas estándar:



Las fichas funcionan como un menú que le permite seleccionar una página en particular.

La buena noticia es que esto funciona muy bien en Android y Windows Mobile 10. Puede desplazarse rápidamente a través de las cabeceras en la parte superior de la pantalla de Android, y pase las páginas reales en Windows 10 móvil.

En iOS, sin embargo, sólo cuatro elementos se muestran, y el Más botón y la elipse no funcionan.

Por otra parte, no hay iconos, y lo que necesita iconos en una TabbedPage para Apple para aprobar la aplicación de la App Store. Mientras que esta instalación de TabbedPage parece ser una manera bastante interesante para generar páginas, no es adecuado para una aplicación multiplataforma. Más adecuado sería el CarouselView, que por desgracia no estaba preparada para el momento en este libro se imprimió.

capítulo 26

diseños personalizados

La personalización es un aspecto crucial de cualquier entorno de programación gráfica. Se ha diseñado el sistema de flexibilidad? ¿Se puede crear nuevos tipos de elementos de interfaz de usuario? ¿Qué tan bien estos nuevos elementos se integran en el sistema?

Xamarin.Forms tiene varias características que facilitan la personalización. Ya has visto el servicio de dependencia (discutido por primera vez en el capítulo 9, "específicas de la plataforma llamadas a la API"), lo que permite su aplicación a ejecutar código específico de la plataforma de la biblioteca de clases portátil común.

En el siguiente capítulo, verá cómo crear elementos de interfaz de usuario especializadas en forma de nuevo Ver derivados. Se crea estos nuevos elementos de codificación personalizada *extracción de grasas* que implementan el elemento en las plataformas individuales.

Este capítulo se centra en cambio en las clases poderosas que normalmente habitan en el árbol visual *Entre la página y los objetos de interfaz de usuario individuales. Estas clases se conocen como diseños porque se derivan de Disposición <Ver>*. Xamarin.Forms define cuatro de tales classes- StackLayout, AbsoluteLayout fuera, Disposición relativa, y Cuadrícula –cada uno de los cuales organiza sus hijos de una manera diferente. Como hemos visto a lo largo de este libro, estos Disposición <Ver> derivados son vitales para la definición de la organización visual de la página.

Las clases que se derivan de Disposición <Ver> son algo inusual en Xamarin.Forms en que hacen uso de ningún código específico de la plataforma. En su lugar, se implementan en su totalidad dentro de Xamarin.Forms.

En este capítulo se describe cómo衍生 una clase de Disposición <Ver> para escribir sus propios diseños personalizados. Esta es una habilidad muy útil que viene al rescate cada vez que necesita para organizar su página de una manera que no está a cargo de las clases de diseño estándar. Por ejemplo, supongamos que desea presentar los datos mediante el uso de una metáfora tarjeta-archivo con tarjetas superpuestas, o como artículos que se envuelven en ambas columnas y filas de desplazamiento, o que Pan de lado a lado con golpes de dedo. Este capítulo le mostrará cómo escribir esas clases.

Escribir diseños personalizados también le ofrece las mejores perspectivas sobre cómo el sistema de diseño en Xamarin.Forms funciona. Este conocimiento le ayudará a diseñar sus propias páginas, incluso si se restringe a sí mismo a las clases de diseño estándar.

Una visión general de diseño

En Xamarin.Forms, no existe un sistema centralizado que gestiona el diseño. En cambio, el proceso es muy descentralizada. Diseño está a cargo de los propios elementos, o dentro de las clases que se derivan de. Por ejemplo, cada elemento visual es responsable de determinar su propio tamaño preferido. Esto se conoce como una

pedido tamaño, ya que puede que no haya espacio suficiente para adaptarse a todo el elemento, o puede haber más que suficiente espacio para el elemento.

Los elementos que consiguen más involucrados en el diseño tienen un solo niño o varios niños. Estos son los

Página derivados, Diseño (derivados ContentView, Marco, y ScrollView), y Disposición <Ver>

derivados. Estos elementos son responsables de determinar la ubicación y el tamaño de su hijo o hijos con respecto a sí mismos. La ubicación y el tamaño del niño por lo general se basan en el tamaño requerido por el niño, por lo que a menudo implica una disposición de dar y llevar la relación entre padres e hijos. Los niños tienen peticiones, pero los padres se imponga la ley.

Veamos algunos ejemplos sencillos.

Padres e hijos

Considere el siguiente código:

```
< Página de contenido ... >
  < Marco OutlineColor = "Acento" >
    < Etiqueta Texto = "Texto de ejemplo" />
  </ Marco >
</ Página de contenido >
```

Es un Etiqueta en un Marco en un Página de contenido. Me gusta más Ver derivados, la Marco tiene por defecto horizontalOptions y VerticalOptions configuración de las propiedades de LayoutOptions.Fill, lo que significa que la Marco llena la página excepción de un posible Relleno establecer en la página. El tamaño de la Marco se basa en el tamaño de la página y no en el tamaño del texto mostrado por el Etiqueta.

Ahora ajuste el HorizontalOptions y VerticalOptions propiedades en el Marco:

```
< Página de contenido ... >
  < Marco OutlineColor = "Acento" 
    VerticalOptions = "Centrar"
    HorizontalOptions = "Centrar" >
    < Etiqueta Texto = "Texto de ejemplo" />
  </ Marco >
</ Página de contenido >
```

los Marco ahora abraza el texto representado de la Etiqueta, lo que significa que el tamaño de la Marco se basa en el tamaño de la Etiqueta en lugar de en el tamaño de la página.

Pero no del todo! Si agrega más y más texto a la Etiqueta, el Marco crecerá, pero no va a llegar nunca mayor que la página. En su lugar, se truncará el texto. Con más texto del que cabe en la página, la Etiqueta se vuelve limitado por el tamaño máximo de la Marco, que está limitada por el tamaño de la EstafatentPage.

Pero ahora poner el Marco en un ScrollView:

```
< Página de contenido ... >
  < ScrollView >
    < Marco OutlineColor = "Acento" >
```

```
< Etiqueta Texto = "Muy largo texto ... " />
</ Marco >
</ ScrollView >
</ Pagina de contenido >
```

Ahora el ScrollView es el tamaño de la página, pero el Marco puede crecer más grande que el ScrollView.

los ScrollView permite al usuario desplazarse a la parte inferior de la Marco a la vista.

los Marco También puede extenderse más allá de la parte inferior de la página si está en una StackLayout:

```
< Pagina de contenido ... >
< StackLayout >
    < Marco OutlineColor = "Acento" >
        < Etiqueta Texto = "Muy largo texto ... " />
    </ Marco >
</ StackLayout >
</ Pagina de contenido >
```

En última instancia, es el padre que determina cuál debe ser el tamaño de sus hijos, e impone que el tamaño de sus hijos, pero a menudo los padres se basará ese tamaño en el tamaño solicitado del niño.

Tamaño y posicionamiento

El proceso de diseño comienza en la parte superior del árbol visual con la página, y luego procede a través de todas las ramas del árbol visual para abarcar todos los elementos visuales de la página. Los elementos que son padres de otros elementos son responsables para el dimensionamiento y la colocación de sus hijos en relación con ellos mismos. Esto requiere que los elementos primarios llaman ciertos métodos públicos en los elementos secundarios. Estos métodos públicos a menudo resultan en llamadas a otros métodos dentro de cada elemento, para las características que se establezcan, y para los eventos a ser despedidos.

Tal vez se llama el método pública más importante que participa en el diseño (muy apropiadamente) **Laico-fuera**. Este método está definido por **VisualElement** y heredado por cada clase que deriva de **VisualElement**:

```
public void Diseño( Rectángulo límites )
```

Los Diseño método especifica dos características del elemento:

- el área rectangular en la que el elemento está mostrada (indicada por la Anchura y Altura propiedades de la Rectángulo valor); y
- la posición de la esquina superior izquierda del elemento con respecto a la esquina superior izquierda de su parent (el X y Y Propiedades).

Cuando una aplicación se pone en marcha y la primera página hay que mostrar, la primera Diseño llamada es a una Página objeto, y el Anchura y Altura propiedades indican el tamaño de la pantalla, o el área de la pantalla que la página ocupa. A partir de esta primera Diseño llama a Diseño las llamadas se propagan de manera efectiva a través del árbol visual: Cada elemento que es un parent a otro elementos- Diseño de página, y Disposición <Ver> derivados es responsable de llamar al Diseño método en sus niños, lo que resulta

en cada elemento visual en la página de recibir una llamada a su Diseño método. (Vas a ver cómo funciona esto en breve).

Todo este proceso se conoce como una *ciclo de diseño*, y si a su vez los lados del teléfono, el ciclo comienza de nuevo diseño desde el principio en la parte superior del árbol visual con el Página objeto. ciclos de diseño también pueden ocurrir en un subconjunto del árbol visual si algo cambia para afectar el diseño. Estos cambios incluyen los artículos que se añaden o se eliminan de una colección tal como el que en una Vista de la lista o una StackLayout

u otro Diseño clase, un cambio en la Es visible propiedad de un elemento, o un cambio en el tamaño de un elemento (por una razón u otra).

interno para VisualElement, el Diseño método hace cinco propiedades del elemento de engaste. Estas propiedades están definidas por VisualElement:

- Límites de tipo Rectángulo
- X de tipo doble
- Y de tipo doble
- Anchura de tipo doble
- Altura de tipo doble

Estas propiedades están todos sincronizados. los X, Y, Ancho, y Altura propiedades de VisualElement son siempre los mismos valores que el X, Y, Ancho, y Altura propiedades de la Límites rectángulo. Estas propiedades indican el tamaño rendido real del elemento y su posición con respecto a la esquina superior izquierda de su parente.

Ninguna de estas cinco propiedades tienen público conjunto descriptores de acceso. Para código externo, estas propiedades son obtenidos solamente.

Antes de un elemento de primera Diseño llama a X y Y propiedades tienen valores de 0, pero el Anchura y Altura propiedades tienen valores "mock" de -1, que indica que las propiedades aún no se han establecido. Los valores válidos de estas propiedades están disponibles sólo después de que se ha producido un ciclo de diseño. Los valores válidos son *no* disponible durante la ejecución de los constructores de los elementos que componen el árbol visual.

Los X, Y, Ancho, y Altura Todas ellas están respaldados por propiedades enlazables, para que puedan ser fuentes de enlaces de datos. Los Límites propiedad es *no* respaldo por una propiedad enlazable y no se dispara una PropertyChanged evento. No utilice Límites como una fuente de enlace de datos.

Una llamada a Diseño También activa una llamada a la SizeAllocated método, que se define por VisualElement al igual que:

```
protected void SizeAllocated ( doble anchura, doble altura)
```

Los dos argumentos son los mismos que la Anchura y Altura propiedades de la Límites rectángulo. los SizeAllocated método llama a un nombre de método virtual protegida OnSizeAllocated:

```
protegido virtual void OnSizeAllocated ( doble anchura, doble altura)
```

Después de la OnSizeAllocated método devuelve, y el tamaño ha cambiado de su valor anterior, VisualElement dispara una SizeChanged evento, que se define así:

```
p\x{00f3;blico evento Controlador de eventos SizeChanged;
```

Esto indica que el tamaño del elemento se ha establecido o que haya cambiado posteriormente. Como hemos visto en cap\x{00f3;culos anteriores, cuando se necesita para implementar un manejo de tama\xf1o espec\xedfico, la SizeChanged evento es una excelente oportunidad de acceder al L\x{00f3;mite la propiedad o la Anchura y Altura propiedades para obtener un tama\xf1o v\x{00e1lido de la p\x{00e1gina o cualquier elemento de la p\x{00e1gina. La llamada a la Dise\xf3;o m\x{00e9todo se completa con el disparo de la SizeChanged evento.

Como alternativa a la SizeChanged evento, es posible que una aplicaci\x{00f3;n para anular OnSizeAllocated situado en un Pagina de contenido derivado para obtener el nuevo tama\xf1o de la p\x{00e1gina. (Si lo hace, aseg\x{00f3;rese de llamar a la implementaci\x{00f3;n de la clase base del OnSizeAllocated). Usted encontrará que OnSizeAllocated se llama a veces, cuando el tama\xf1o del elemento no cambia realmente. Los SizeChanged evento se activa s\x{00f3;lo cuando los cambios de tama\xf1o, y es mejor para el manejo de tama\xf1o espec\xedfico en el nivel de aplicaci\x{00f3;n.

Los OnSizeAllocated m\x{00e9todo no se define como virtual, de modo que las aplicaciones pueden anular, pero para permitir que las clases dentro de Xamarin.Forms para anularlo. S\x{00f3;lo dos clases de anulaci\x{00f3;n OnSizeAllocated para llevar a cabo su propio procesamiento especializado, pero son clases de excepcional importancia:

- Página
- Diseño

Estas son las clases base para todos los elementos Xamarin.Forms que sirven como padres a otros elementos dentro de un \x{00e1rbol visual Xamarin.Forms. (A pesar de que Vista de la lista y TableView parecen tener hijos, as\x{00f3;, la disposici\x{00f3;n de esos ni\xf3os se manejan dentro de las implementaciones de la plataforma de estos puntos de vista.)

Algunas de las clases que se derivan de Página y Diseño tener un Contenido propiedad de tipo View. Estas clases son Pagina de contenido, ContentView, Marco, y ScrollView. Los Contenido la propiedad es un solo ni\xf3o. Las otras clases que se derivan de Página (MasterDetailPage, TabbedPage, y CarouselPage) tener varios hijos. Las clases que se derivan de Disposici\x{00f3;n <View> tener un Ni\xf3os propiedad de tipo IList<View>; estas clases son StackLayout, AbsoluteLayout, RelativeLayout, y

Cuadricula.

Los Página y Diseño clases tienen una estructura paralela a partir de una anulación de la OnSizeAllocated m\x{00e9todo. Ambas clases definen el siguiente m\x{00e9todo que se llama desde el OnSizeAllocated anular:

```
protected void UpdateChildrenLayout ()
```

Ambas versiones de UpdateChildrenLayout llamar a un m\x{00e9todo llamado LayoutChildren. Este m\x{00e9todo se define s\x{00f3;lo un poco diferente en Página y Diseño. En Página, el layoutChildren m\x{00e9todo se define como virtual:

```
protegido virtual void layoutChildren ( doble X, doble Y, doble anchura, doble altura)
```

En Diseño se define como el resumen:

```
protegido abstract void layoutChildren ( doble X, doble Y, doble anchura, doble altura);
```

Cada clase Xamarin.Forms que tiene un Contenido o una Niños la propiedad también tiene una reemplazable layoutChildren método. Cuando usted escribe su propia clase que deriva de Disposición <Ver> (que es el objetivo principal de este capítulo), podrás anular layoutChildren para proporcionar una organización personalizada de los niños del diseño.

La responsabilidad de una layoutChildren la redefinición es llamar a la Diseño método en todos los hijos del elemento, que suele ser el Ver Fije el objeto al elemento de Contenido la propiedad o la Ver objetos en el elemento de Niños colección. Esta es la parte más importante de diseño.

Como se recordará, una llamada a la Diseño método da como resultado la Límites, X, Y, Ancho, y Altura propiedades que se establecen y en las llamadas a SizeAllocated y OnSizeAllocated. Si el elemento es una Diseño derivado, entonces OnSizeAllocated llamadas UpdateChildrenLayout y LayoutChildren. Diseño-Niños luego llama Diseño en sus hijos. Así es como el Diseño las llamadas se propagan desde la parte superior del árbol visual a través de todas las ramas y cada elemento de la página.

Ambos Página y Diseño también definir una LayoutChanged evento:

```
evento público Controlador de eventos LayoutChanged;
```

los UpdateChildrenLayout método concluye por el disparo de este evento, pero sólo si al menos un hijo tiene un nuevo Límites propiedad.

Usted ha visto que la Página y Diseño clases tanto anulan la OnSizeAllocated método, y ambos definir UpdateChildrenLayout y layoutChildren métodos y una LayoutChanged evento. los Página y Diseño clases tienen otra similitud: Ambos definen una Relleno propiedad. Este relleno se refleja automáticamente en los argumentos LayoutChildren.

Por ejemplo, considere la siguiente definición página:

```
< Pagina de contenido ... Relleno = " 20 ">
  < ContentView Relleno = " 15 ">
    < Etiqueta Texto = " Texto de ejemplo " />
  </ ContentView >
</ Pagina de contenido >
```

Supongamos que la pantalla en modo vertical mide 360 por 640. La Pagina de contenido recibe una llamada a su Diseño método con un rectángulo límites igual a (0, 0, 360, 640). Esto inicia el ciclo de diseño.

Aunque el Diseño método en el Pagina de contenido tiene un argumento de (0, 0, 360, 640), la Diseño-Niños llamar en esa página se ajusta para el Relleno propiedad de 20. Tanto el anchura y altura se redujo en 40 (20 en cada lado) y la X y y argumentos se incrementan en 20, por lo que la Laico-outChildren argumentos son (20, 20, 320, 600). Este es el rectángulo respecto a la página en la que EstatentPage puede posicionar su hijo.

los layoutChildren método en el Pagina de contenido llama a la Diseño método en su hijo (el Estaf-
tentView) para dar la ContentView todo el espacio disponible a la página menos el relleno en la página. El argumento límites del rectángulo a
este Diseño llamada es (20, 20, 320, 600), que posiciona la esquina superior izquierda de la ContentView 20 unidades a la derecha y por
debajo de la esquina superior izquierda de la Estafa-
tentPage.

La llamada a la layoutChildren SOBREPASO ContentView refleja que el área de diseño, pero disminuyó por la Relleno ajuste de
15, por lo que los argumentos de la layoutChildren SOBREPASO ContentView son (15, 15, 290, 570). Esta layoutChildren llama al
método Diseño método en el Etiqueta con ese valor.

Ahora vamos a hacer un pequeño cambio:

```
< Pagina de contenido ... Relleno = " 20 " >
  < ContentView Relleno = " 15 " >
    VerticalOptions = " Centrar " >
      < Etiqueta Texto = " Texto de ejemplo " />
    </ ContentView >
</ Pagina de contenido >
```

los layoutChildren SOBREPASO Pagina de contenido Ahora tiene que hacer las cosas un poco diferente. No puede simplemente llamar Diseño sobre
el ContentView con su propio tamaño, menos el relleno. Se debe llamar al Diseño
método en el ContentView a verticalmente centrar la ContentView dentro del espacio que tiene disponible.

¿Pero cómo? Para centrar verticalmente la ContentView con respecto a sí mismo, el Pagina de contenido debe conocer la altura de la ContentView.
Sin embargo, la altura de la ContentView depende de la altura de la Etiqueta,
y que depende de la altura del texto y tal vez en varias propiedades de fuentes que puedan establecerse en el
Etiqueta. Por otra parte, la Etiqueta es capaz de ajustar el texto a varias filas, y el Etiqueta no se puede averiguar cuántas filas se requiere sin
conocer también el espacio horizontal que está disponible para él.

Este problema implica que más pasos están involucrados.

Restricciones de tamaño y solicitudes

Que acaba de ver cómo, en algunos casos, una layoutChildren anulación puede llamar Diseño en su niño o los niños basándose únicamente en
el layoutChildren argumentos. Sin embargo, en el caso más general, layoutChildren
necesita saber el tamaño de sus hijos antes de llamar a los de los niños Diseño métodos. Por esta razón, una layoutChildren anular las
llamadas en general *dos* métodos públicos en este orden en cada uno de sus hijos:

- GetSizeRequest
- Diseño

¿Por qué necesita un padre para llamar GetSizeRequest en su hijo? ¿Por qué no puede el padre simplemente obtener el tamaño del niño
accediendo al niño de Límites propiedad o su Anchura y Altura propiedades?

Debido a que, en el caso general, estas propiedades no se han fijado todavía! Hay que recordar que estas propiedades se establecen mediante una
llamada a Diseño, y el Diseño aún no se ha producido la llamada. En el caso general, la Diseño llamada no puede ocurrir hasta que el padre sabe el tamaño
requerido por el niño. En el caso general, la GetSizeRequest

llamada es un requisito previo para la Diseño llamada.

La información que GetSizeRequest retorna es totalmente independiente de cualquier información que pueda ser ajustado con Diseño. En su lugar, el argumento de Diseño por lo general depende de la información que devuelve GetSizeRequest.

los GetSizeRequest llamada obtiene lo que se llama a veces un *deseado* tamaño de un elemento. Esto a menudo se relaciona con el elemento de *tamaño nativo*, y que generalmente depende de la plataforma en particular. En contraste, el Diseño llamada impone un tamaño determinado en el elemento. A veces, estos dos tamaños son los mismos y otras veces no. Estas dos medidas son por lo general *no lo mismo* si el elemento de HorizontalOpti

y VerticalOptions ajustes están LayoutOptions.Fill. En ese caso, el tamaño que ocupa el elemento se basa generalmente en el área disponible para el padre del elemento en lugar del tamaño que las necesidades de elementos.

El tamaño natural de algunos elementos es fija e inflexible. Por ejemplo, en cualquier plataforma en particular, una Cambiar es siempre un tamaño fijo determinado por su implementación en esa plataforma. Pero eso no es siempre el caso para otros tipos de elementos. A veces una dimensión del tamaño es fija, pero la otra dimensión es más flexible. La altura de una horizontal deslizador se fija por la implementación de la plataforma, pero la anchura de la deslizador puede ser tan amplia como su padre.

A veces el tamaño de un elemento depende de su configuración de propiedades. El tamaño de una Botón o Etiqueta depende del texto mostrado por el elemento y el tamaño de la fuente. Debido a que el texto que se muestra por una Etiqueta puede envolver a varias líneas, la altura de una Etiqueta depende de cuántas filas se muestran, y eso es gobernado por la anchura disponible para el Etiqueta. A veces, la altura o la anchura de un elemento depende de la altura o la anchura de sus hijos. Tal es el caso con StackLayout.

Estas complicaciones requieren que un elemento a determinar su tamaño basado en *restricciones*, que generalmente indican la cantidad de espacio disponible dentro de la matriz del elemento para ese elemento.

Me gusta Diseño, el GetSizeRequest método está definido por VisualElement. Este es un método público que un elemento padre llama para obtener el tamaño de cada uno de sus hijos:

```
pùblico virtual SizeRequest GetSizeRequest ( doble widthConstraint, doble heightConstraint)
```

los widthConstraint y heightConstraint argumentos generalmente indican el tamaño que el padre tiene disponible para el niño; el niño es responsable de implementar este método para determinar un tamaño adecuado para sí mismo sobre la base de esas limitaciones. Por ejemplo, una Etiqueta determina el número de líneas que necesita para su texto sobre la base de una anchura determinada.

VisualElement también define un método protegido muy similar llamado OnSizeRequest:

```
protegido virtual SizeRequest OnSizeRequest ( doble widthConstraint, doble heightConstraint)
```

Obviamente, estos dos métodos están relacionados y son fácilmente confundidos. Ambos métodos se definen como virtual, pero a lo largo de toda Xamarin.Forms, sólo una clase anula la GetSizeRequest método, y esa es la Diseño clase, que marca el procedimiento que se sellado.

Por otro lado, cada clase que deriva de Diseño o Disposición <Ver> anulaciones OnSizeRequest. Aquí es donde una clase de diseño determina el tamaño que necesita estar haciendo llamadas a la GetSizeRequest métodos de sus hijos.

por Ver derivados de (pero no Diseño derivados), el público GetSizeRequest llama al método protegido OnSizeRequest método que es responsable de obtener el tamaño nativo del elemento de la aplicación específica de la plataforma.

los SizeRequest estructura regresó de GetSizeRequest y OnSizeRequest tiene dos propiedades:

- Solicitud de tipo tamaño
- Mínimo de tipo tamaño

Es tentador tratar de llamar GetSizeRequest en los objetos de nueva creación, como Etiqueta y BoxView y deslizador, y examinar qué tamaños son devueltos. sin embargo, el GetSizeRequest llamada no funcionará a menos que el elemento es parte de un árbol visual real, porque sólo entonces es el elemento Xamarin.Forms implementado con un objeto plataforma subyacente.

La mayoría de los elementos de retorno SizeRequest valores con idéntica Solicitud y Mínimo tamaños. Los únicos elementos para los cuales están uniformemente es diferente Vista de la lista y TableView, donde el Mínimo tamaño es (40, 40), tal vez para permitir que una parte de la Vista de la lista o TableView que se mostrará incluso si no hay suficiente espacio para toda la cosa.

En general, sin embargo, el Mínimo el tamaño no parecen jugar un gran papel en el sistema Xamarin.Forms diseño, y no es necesario ir a medidas extraordinarias para acomodarlo. los Tamaño-

Solicitud la estructura tiene un constructor que le permite configurar las propiedades de la misma tamaño valor.

Se puede recordar que VisualElement define cuatro propiedades que tienen la palabra Solicitud como parte de sus nombres:

- WidthRequest de tipo doble
- HeightRequest de tipo doble
- MinimumWidthRequest de tipo doble
- MinimumHeightRequest de tipo doble

A diferencia de la Anchura y Altura propiedades, estos cuatro establecimientos públicos conjunto descriptores de acceso. Su aplicación puede establecer la WidthRequest y HeightRequest propiedades de un elemento para anular su tamaño habitual. Esto es particularmente útil para una BoxView, que inicializa su WidthRequest y Altura-

Solicitud los valores a 40. Se puede establecer estas propiedades a valores diferentes para hacer una BoxView cualquier tamaño que deseé.

Por defecto, estas cuatro propiedades tienen valores de "falsas" de -1. Si se ajustan a los valores reales, así es como GetSizeRequest y OnSizeRequest interactuar con ellos:

Primero, GetSizeRequest se encuentra el mínimo de su widthConstraint argumento y el elemento de WidthRequest la propiedad y el mínimo de heightConstraint y HeightRequest. Estos son los valores pasados a OnSizeRequest. En esencia, el elemento se ofrece únicamente la cantidad de tamaño que el WidthRequest y HeightRequest propiedades indican.

Sobre la base de esas limitaciones, OnSizeRequest devuelve una SizeRequest valor de nuevo a GetSize-Solicitud. Ese SizeRequest tiene valor Solicitud y Mínimo propiedades. GetSizeRequest luego encuentra el mínimo de la Anchura y Altura propiedades de la Solicitud propiedad y el WidthRe-búsqueda y HeightRequest propiedades fijan en el elemento. También encuentra el mínimo de la Anchura y Altura propiedades de la Mínimo propiedad, y el MinimumWidthRequest y MinimumHeightRequest propiedades fijan en el elemento. GetSizeRequest a continuación, devuelve un nuevo SizeRequest valor en función de estos mínimos.

He aquí alguna simple de marcas:

```
< Pagina de contenido ... Relleno = " 20 " >
  < Etiqueta Texto = " Texto de ejemplo "
    HorizontalOptions = " Centrar "
    VerticalOptions = " Centrar " />
</ Pagina de contenido >
```

Supongamos que la pantalla en modo vertical es de 360 por 640. El ciclo de diseño comienza con una llamada a la Lai- fuera método de Pagina de contenido con un rectángulo límites de (0, 0, 360, 640). Los argumentos de la Lai- outChildren SOBREPASO Pagina de contenido son ajustados por el relleno, por lo que los argumentos son (20, 20, 320, 600).

Porque Etiqueta tiene su HorizontalOptions y VerticalOptions propiedades no establecida Lai- outOptions.Fill, la página debe determinar el tamaño de la Etiqueta llamando GetSizeRequest con las limitaciones de (320, 600). La información que Etiqueta rendimientos dependen de la plataforma, pero asumamos que el Etiqueta devuelve un tamaño de (100, 24). Los Pagina de contenido debe entonces la posición que Etiqueta en el centro de la zona (320, 600) disponible para su niño. A partir de la anchura de 320, se resta el Etiqueta ancho de 100 y se divide por 2. Eso es 110, pero eso es en relación con el área disponible para el niño, y no en relación con la esquina superior izquierda de la página, que incluye el margen de 20. Así que el desplazamiento de la horizontal Etiqueta desde el Pagina de contenido es en realidad 130.

los Pagina de contenido realiza un cálculo similar para la altura: 600 menos 24, dividido por 2, además de 20, o 308. La Pagina de contenido luego llama al Diseño método de la Etiqueta Con el rectángulo límites (130, 308, 100, 24) en la posición y el tamaño de la Etiqueta con respecto a sí mismo.

Como hacer WidthRequest y HeightRequest configuración de la Etiqueta afectará esto? Aquí está un WidthRe-búsqueda es decir más de lo que el Etiqueta necesidades, sino una HeightRequest que es menor:

```
< Etiqueta Texto = " Texto de ejemplo "
  WidthRequest = " 200 "
  HeightRequest = " 12 "
  HorizontalOptions = " Centrar "
  VerticalOptions = " Centrar " />
```

los Pagina de contenido todavía llama a la GetSizeRequest método de la Etiqueta con las limitaciones de (320, 600), pero la GetSizeRequest modifica las restricciones para ser (200, 12), y eso es lo que se pasa a la OnSizeRequest anular. los Etiqueta todavía devuelve un tamaño solicitado de (100, 24), pero GetSizeRequest nuevamente ajusta los de la Anchura y Altura solicitud y devuelve (200, 12) de nuevo a la Pagina de contenido.

los Pagina de contenido luego llama al Diseño método de Etiqueta Residencia en Etiqueta dimensiones de (200, 12) en lugar de (100, 24). los Diseño obtener de la Etiqueta ahora tiene un rectángulo límites de (80, 314, 200, 12). los Etiqueta se muestra con el doble de anchura que es necesaria para el texto, pero con la mitad de la altura. El texto se recorta por el fondo.

Si en lugar del WidthRequest el establecimiento de la Etiqueta está dispuesto a ser de menos de 100, por ejemplo, 50- entonces la OnSizeRequest método se llama con una widthConstraint argumento de 50, y el Etiqueta calcula una altura para el texto que se traduce en envolver el texto en varias líneas.

limitaciones infinitas

Ahora, aquí hay algunas marcas que a primera vista parece muy similar al ejemplo anterior, pero con una diferencia bastante profunda:

```
< Pagina de contenido ... Relleno = " 20 " >
  < StackLayout >
    < Etiqueta Texto = " Texto de ejemplo " />
    ...
  </ StackLayout >
</ Pagina de contenido >
```

los Pagina de contenido todavía recibe una inicial Diseño llamar con los argumentos (0, 0, 360, 640), y los argumentos de la layoutChildren override son (20, 20, 320, 600). Tiene un hijo, el StackLayout. los StackLayout posee una configuración predeterminada de HorizontalOptions y VerticalOptions de LayoutOptions.Fill, lo que significa que la StackLayout puede ser posicionado en relación con Pagina de contenido con un Diseño llamar de (20, 20, 320, 600).

Esto resulta en StackLayout conseguir una layoutChildren llamar con argumentos de (0, 0, 320, 600). Cómo StackLayout tamaño y la posición de sus hijos?

Como sabemos por el trabajo con StackLayout desde el capítulo 4, una línea vertical StackLayout da a sus hijos el mismo tamaño horizontal como en sí, sino un tamaño vertical en base a lo que el niño necesita. Esto significa que StackLayout debe llamar GetSizeRequest en todos sus hijos antes de llamar Diseño. Pero, ¿qué limitaciones se debe especificar con las GetSizeRequest llamadas?

El impulso inicial podría ser que StackLayout llamadas GetSizeRequest en sus niños con limitaciones que reflejan su propio tamaño de (320, 600). Pero eso no es correcto. los StackLayout no limita sus hijos a su propia altura. Permite que sus hijos sean cualquier altura que necesitan ser. Esto implica que la restricción de altura debe ser en realidad infinita.

Y esto es cierto. StackLayout llamadas GetSizeRequest en sus niños con una altura de (320, ∞), o, en términos de .NET, (320, Double.PositiveInfinity).

Esto es importante: Limitaciones pasaron a GetSizeRequest y OnSizeRequest puede variar desde 0 a través Double.PositiveInfinity. Sin embargo, GetSizeRequest y OnSizeRequest puede mismos nunca solicitar una dimensión infinita, devolviendo una SizeRequest valor con un conjunto de propiedades a Double.PositiveInfinity.

Vamos a tratar de otro patrón de diseño común:

```
< Pagina de contenido ... Relleno = " 20 " >
    < ScrollView >
        < StackLayout >
            < Etiqueta Texto = " Texto de ejemplo " />
            ...
        </ StackLayout >
    </ ScrollView >
</ Pagina de contenido >
```

Como siempre, Pagina de contenido recibe una llamada a Diseño con un rectángulo límites de (0, 0, 360, 640) y una llamada a su layoutChildren método con argumentos de (20, 20, 320, 600). los ScrollView tiene por defecto HorizontalOptions y VerticalOptions ajustes de LayoutOptions.Fill, por lo que la página no necesita saber el tamaño de la ScrollView es. La página llama simplemente el Diseño método de ScrollView con un rectángulo límites de (20, 20, 320, 600).

ScrollView luego se hace una llamada a su layoutChildren método con argumentos de (0, 0, 320, 600). Se necesita determinar el tamaño de su hijo (el StackLayout), por lo que llama a la GetSizeRequest método de StackLayout. ¿Cuáles deberían ser las limitaciones?

En el caso general, la StackLayout tendrán una altura mayor que la altura de ScrollView. Es por eso que está incluida una ScrollView en el árbol visual! ScrollView tiene que saber que la altura si se va a desplazarse con éxito su hijo. Por lo tanto, ScrollView llama a la GetSizeRequest método de StackLayout con las limitaciones de (320, Double.PositiveInfinity). Esto se traduce en una llamada a En-SizeRequest con los mismos argumentos de restricción, los cuales StackLayout anulaciones y mangos.

También se puede pensar en una restricción infinito como una indicación de tamaño automático. Un vertical StackLayout pide un tamaño niño con una restricción de altura infinita para obtener la altura requerida del niño. Del mismo modo, un niño de un Cuadrícula células cuya altura fila o ancho de la columna es GridLength.Auto verá un infinito heightConstraint o widthConstraint, o ambos. Un niño de una AbsoluteLayout con un Diseño-Límites altura o anchura de Auto También verá un infinito heightConstraint o widthConstraint.

A veces las palabras *constreñido* y *sin restricciones* se usan para referirse a estas diferencias. Un elemento es *constreñido* cuando recibe una llamada a su GetSizeRequest método con argumentos noninfinite. El elemento está limitado a un tamaño particular. Un elemento es *sin restricciones* cuando se hace una llamada a Obtener-SizeRequest con uno o ambos argumentos igual a Double.PositiveInfinity. A veces el término *parcialmente restringido* se utiliza para referirse a una GetSizeRequest llamar con una sola Double.PositiveIn-finity argumento, y el término *totalmente limitado* deja claro que ni el argumento es infinito.

Cuando usted escribe sus propias clases de diseño personalizados mediante la derivación de Disposición <Ver>, debe invalidar tanto la OnSizeRequest y layoutChildren métodos, y hay que tener en cuenta que en virtud de

certas circunstancias, uno o ambos de los argumentos de las limitaciones OnSizeRequest estarán duplicados con ble.PositiveInfinity. Sin embargo, OnSizeRequest Nunca debe solicitar un tamaño infinito.

El mirar a escondidas dentro del proceso

Gran parte de la información presentada hasta ahora en este capítulo se ha montado a partir de programas de prueba que contienen clases que se derivan de diversos elementos (tales como StackLayout, ScrollView, y Etiqueta), reemplazar los métodos virtuales (tales como GetSizeRequest, OnSizeRequest, OnSizeAllocated, y Laico-outChildren), y simplemente mostrar información en el Salida ventana de Visual Studio o Xamarin Studio mediante el Debug.WriteLine método de la Diagnóstico del sistema espacio de nombres.

Un poco de ese proceso de exploración, pero utilizando el propio teléfono para mostrar esta información, se muestra en el **ExploreChildSizes** muestra.

ExploreChildSizes utiliza una MasterDetailPage para mostrar un montón de botones de radio en el Dominar página y un árbol visual en el Detalle parte. Los botones de radio hacen uso de la RadioButtonManager y RadioButtonItem clases presentados en el capítulo 25, "variedades de página." Aquí está el Dominar página con los botones de radio para seleccionar HorizontalOptions y VerticalOptions propiedades de los puntos de vista del niño en el Detalle página:

```
< MasterDetailPage xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    xmlns: locales = "clr-espacio de nombres: ExploreChildSizes; montaje = ExploreChildSizes "
    xmlns: kit de herramientas =
    "clr-espacio de nombres: Xamarin.FormsBook.Toolkit; montaje = Xamarin.FormsBook.Toolkit "
    x: Class = " ExploreChildSizes.ExploreChildSizesPage " >

< MasterDetailPage.Master >
    < Página de contenido Título = " intercambiar " >
        < ContentPage.Icon >
            < OnPlatform x: TypeArguments = " FileImageSource "
                WinPhone = " Imágenes / refresh.png " />
        </ ContentPage.Icon >

        < ContentPage.Padding >
            < OnPlatform x: TypeArguments = " Espesor "
                iOS = " 0, 20, 0, 0 " />
        </ ContentPage.Padding >

        < ScrollView >
            < StackLayout Relleno = " 20 "
                Especificado = " 20 " >

                < StackLayout >
                    < StackLayout.BindingContext >
                        < kit de herramientas: RadioButtonManager x: Nombre = " vertRadios "
                            x: TypeArguments = " LayoutOptions " />
                    </ StackLayout.BindingContext >

                    < StackLayout HorizontalOptions = " comienzo " >
                        < Etiqueta Texto = " VerticalOptions niño " >
```

```

        Tamaño de fuente = " Medio " />
    < BoxView Color = " Acento "
        HeightRequest = " 3 " />
    </ StackLayout >

    < locales: RadioButton BindingContext = "{Binding Articulos [0]} " />
    < locales: RadioButton BindingContext = "{Binding Articulos [1]} " />
    < locales: RadioButton BindingContext = "{Binding Articulos [2]} " />
    < locales: RadioButton BindingContext = "{Binding Articulos [3]} " />
    < locales: RadioButton BindingContext = "{Binding Articulos [4]} " />
    < locales: RadioButton BindingContext = "{Binding Articulos [5]} " />
    < locales: RadioButton BindingContext = "{Binding Articulos [6]} " />
    < locales: RadioButton BindingContext = "{Binding Articulos [7]} " />
</ StackLayout >

< StackLayout >
    < StackLayout.BindingContext >
        < kit de herramientas: RadioButtonManager x: Nombre = " horzRadios "
            x: TypeArguments = " LayoutOptions " />
    </ StackLayout.BindingContext >

    < StackLayout HorizontalOptions = " comienzo " >
        < Etiqueta Texto = " HorizontalOptions niño "
            Tamaño de fuente = " Medio " />
        < BoxView Color = " Acento "
            HeightRequest = " 3 " />
    </ StackLayout >

    < locales: RadioButton BindingContext = "{Binding Articulos [0]} " />
    < locales: RadioButton BindingContext = "{Binding Articulos [1]} " />
    < locales: RadioButton BindingContext = "{Binding Articulos [2]} " />
    < locales: RadioButton BindingContext = "{Binding Articulos [3]} " />
    < locales: RadioButton BindingContext = "{Binding Articulos [4]} " />
    < locales: RadioButton BindingContext = "{Binding Articulos [5]} " />
    < locales: RadioButton BindingContext = "{Binding Articulos [6]} " />
    < locales: RadioButton BindingContext = "{Binding Articulos [7]} " />
</ StackLayout >
</ StackLayout >
</ ScrollView >
</ Pagina de contenido >
</ MasterDetailPage.Master >
...
</ MasterDetailPage >
```

Esta página utiliza una clase llamada `RadioButtonManager` en el `Xamarin.FormsBook.Toolkit` biblioteca, que se puede leer en su tiempo libre. Permite por ser una fuente de unión para un elemento asociado con el botón seleccionado. Los Botón de radio clase utiliza la `Acento` el color y la `Negrita` atributo para indicar el elemento seleccionado:

```

< ContentView xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x: Class = " ExploreChildSizes.RadioButton " >

    < Etiqueta Texto = "{Nombre} Encuadernación "
        Tamaño de fuente = " Medio " >
```

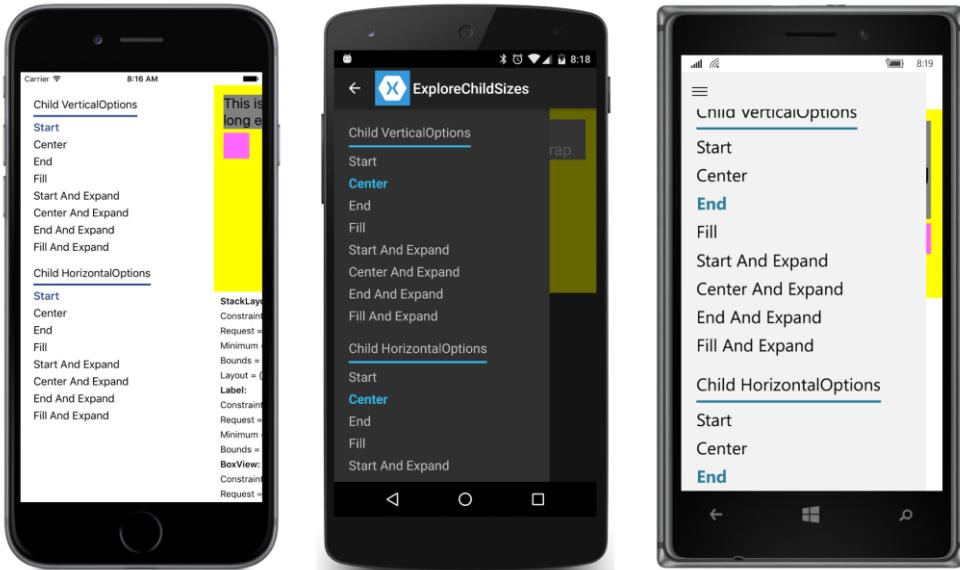
```

< Label.GestureRecognizers >
    < TapGestureRecognizer Mando = "0 Comando Encuadernación" 
        CommandParameter = "0 Valor Encuadernación" />
</ Label.GestureRecognizers >

< Label.Triggers >
    < DataTrigger Tipo de objetivo = "Etiqueta" 
        Unión = "(Binding) IsSelected" 
        Valor = "Cierto" >
        < Setter Propiedad = "Color de texto" Valor = "Acento" />
        < Setter Propiedad = "FontAttributes" Valor = "Negrita" />
    </ DataTrigger >
</ Label.Triggers >
</ Etiqueta >
</ ContentView >

```

Aquí está la Dominar página en las tres plataformas. En el lado derecho de las tres pantallas, se puede ver una parte del Detalle página con un fondo amarillo de un StackLayout:



los Detalle página (que se muestra a continuación) se divide por una rejilla en dos filas de igual altura. La fila superior es un árbol visual simple que consiste en una StackLayout con un Etiqueta y BoxView. Sin embargo, las clases de este árbol visual son en realidad *derivado* de StackLayout, Etiqueta, y BoxView y se llaman Abierto-StackLayout, Abierto, y OpenBoxView. Observe que el VerticalOptions y Horizontal-options propiedades de Abierto y OpenBoxView están unidos a los dos RadioButtonManager objetos de la Dominar página:

```

< MasterDetailPage ...>
...
< MasterDetailPage.Detail >
    < Pagina de contenido >

```

```

< ContentPage.Padding >
    < OnPlatform x: TypeArguments = " Espesor "
        iOS = " 0, 20, 0, 0 " />
</ ContentPage.Padding >

< Cuadricula >
    < locales: OpenStackLayout x: Nombre = " openStackLayout "
        Grid.Row = " 0 "
        Color de fondo = " Amarillo "
        Relleno = " 15 " >
        < locales: OpenLabel
            x: Nombre = " abierto "
            Texto = " Esta es una etiqueta con el texto suficientemente tiempo suficiente para envolver "
            Tamaño de fuente = " Grande "
            Color de fondo = " gris "
            VerticalOptions = " {Binding Fuente = {x: vertRadios Referencia},
                Path = SelectedValue} "
            HorizontalOptions = " {Binding Fuente = {x: horzRadios Referencia},
                Path = SelectedValue} " />
        < locales: OpenBoxView
            x: Nombre = " openBoxView "
            Color = " Rosado "
            VerticalOptions = " {Binding Fuente = {x: vertRadios Referencia},
                Path = SelectedValue} "
            HorizontalOptions = " {Binding Fuente = {x: horzRadios Referencia},
                Path = SelectedValue} " />
        </ locales: OpenStackLayout >
        ...
    </ Cuadricula >
</ Pagina de contenido >
</ MasterDetailPage.Detail >
</ MasterDetailPage >

```

los Abierto prefijo en este contexto significa que estas clases definen las propiedades públicas que revelan los argumentos y los valores de la devolución GetSizeRequest llama y (en el caso de OpenStackLayout) los argumentos a LayoutChildren. Todas estas propiedades están respaldados por sólo lectura propiedades enlazables para que puedan servir como fuentes de enlaces de datos. además, el Límites la propiedad se refleja en una propiedad denominada ElementBounds. También el respaldo de una propiedad enlazable de sólo lectura:

Aquí esta la Abierto clase. Los otros dos son similar:

```

clase Abierto : Etiqueta
{
    estático solo lectura BindablePropertyKey ConstraintKey =
        BindableProperty .CreateReadOnly (
            "Restricción",
            tipo de ( tamaño ),
            tipo de ( Abierto ),
            nuevo tamaño ());
}

sólo lectura estática pública BindableProperty ConstraintProperty =
    ConstraintKey.BindableProperty;

```

```
estático solo lectura BindablePropertyKey SizeRequestKey =
BindableProperty.CreateReadOnly (
    "SizeRequest",
    tipo de ( SizeRequest ),
    tipo de ( Abierto ),
    nuevo SizeRequest () );

público estático solo lectura BindableProperty SizeRequestProperty =
SizeRequestKey.BindableProperty;

estático solo lectura BindablePropertyKey ElementBoundsKey =
BindableProperty.CreateReadOnly (
    "ElementBounds",
    tipo de ( Rectángulo ),
    tipo de ( Abierto ),
    nuevo Rectángulo ());

público estático solo lectura BindableProperty ElementBoundsProperty =
ElementBoundsKey.BindableProperty;

público Abierto()
{
    SizeChanged += (remitente, args) =>
    {
        ElementBounds = Límites;
    };
}

público tamaño Restricción
{
    conjunto privado {EstablecerValor (ConstraintKey, valor );}
    obtener {regreso ( tamaño ) GetValue (ConstraintProperty);}
}

público SizeRequest SizeRequest
{
    conjunto privado {EstablecerValor (SizeRequestKey, valor );}
    obtener {regreso ( SizeRequest ) GetValue (SizeRequestProperty);}
}

público Rectángulo ElementBounds
{
    conjunto privado {EstablecerValor (ElementBoundsKey, valor );}
    obtener {regreso ( Rectángulo ) GetValue (ElementBoundsProperty);}
}

público anular SizeRequest GetSizeRequest ( doble widthConstraint, doble heightConstraint)
{
    restricción = nuevo tamaño (WidthConstraint, heightConstraint);
    SizeRequest sizeRequest = base .GetSizeRequest (widthConstraint, heightConstraint);
    SizeRequest = sizeRequest;
    regreso sizeRequest;
}

}
```

La mitad inferior de la Cuadrícula sobre el Detalle página contiene un desplazable StackLayout con enlaces de datos para mostrar estas propiedades:

```
< MasterDetailPage ... >
...
< MasterDetailPage.Detail >
< Pagina de contenido >
< ContentPage.Padding >
< OnPlatform x: TypeArguments = " Espesor "
    iOS = " 0, 20, 0, 0 " />
</ ContentPage.Padding >

< Cuadrícula >
...
< ScrollView Grid.Row = " 1 "
    Relleno = " 10, 0 " >
< StackLayout >
< StackLayout.Resources >
< ResourceDictionary >
< Estilo Tipo de objetivo = " Etiqueta " >
< Setter Propiedad = " Tamaño de fuente " Valor = " Pequeña " />
</ Estilo >
</ ResourceDictionary >
</ StackLayout.Resources >

< StackLayout >
    BindingContext = " {Binding Fuente = {x: Referencia openStackLayout} } "
    < Etiqueta Texto = " StackLayout: "
        FontAttributes = " Negrita " />
    < Etiqueta Texto = " {Binding Path = restricción,
        StringFormat = 'Restricción = {0}' } " />
    < Etiqueta Texto = " {Binding Path = SizeRequest.Request,
        StringFormat = 'Solicitud = {0}' } " />
    < Etiqueta Texto = " {Binding Path = SizeRequest.Minimum,
        StringFormat = 'Mínimo = {0}' } " />
    < Etiqueta Texto = " {Binding Path = ElementBounds,
        StringFormat = 'Límites = {0}' } " />
    < Etiqueta Texto = " {Binding Path = LayoutBounds,
        StringFormat = 'Layout = {0}' } " />
</ StackLayout >
< StackLayout BindingContext = " {Binding Fuente = {x: Referencia openLabel} } "
    < Etiqueta Texto = " Etiqueta: "
        FontAttributes = " Negrita " />
    < Etiqueta Texto = " {Binding Path = restricción,
        StringFormat = 'Restricción = {0}' } " />
    < Etiqueta Texto = " {Binding Path = SizeRequest.Request,
        StringFormat = 'Solicitud = {0}' } " />
```

```
StringFormat = 'Solicitud = {0}' " />

< Etiqueta Texto = " {Binding Path = SizeRequest.Minimum,
StringFormat = 'Mínimo = {0}' } " />

< Etiqueta Texto = " {Binding Path = ElementBounds,
StringFormat = 'Límites = {0}' } " />
</ StackLayout >

< StackLayout BindingContext = " {Binding Fuente = {x: Referencia openBoxView} } " >
< Etiqueta Texto = " BoxView: "
FontAttributes = " Negrita " />

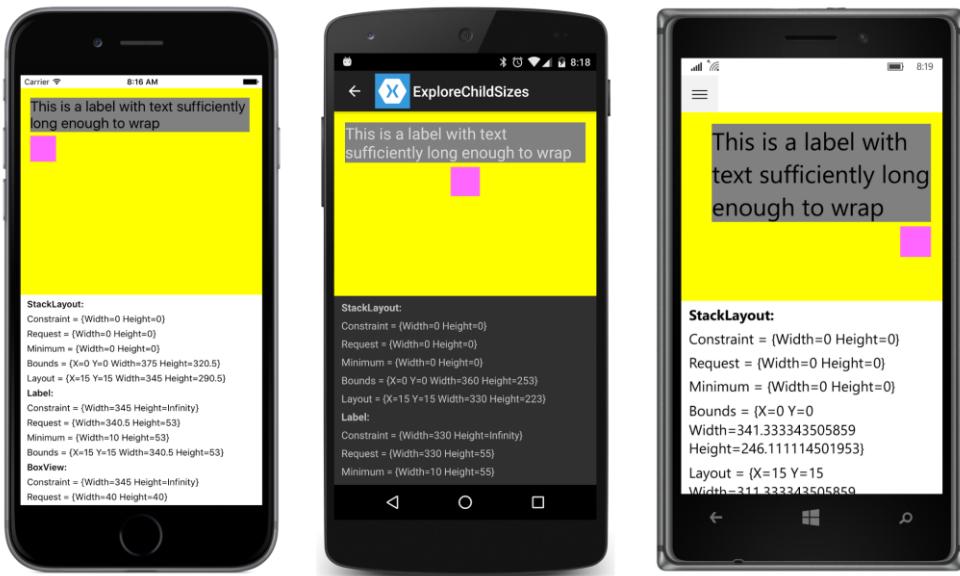
< Etiqueta Texto = " {Binding Path = restricción,
StringFormat = 'Restricción = {0}' } " />

< Etiqueta Texto = " {Binding Path = SizeRequest.Request,
StringFormat = 'Solicitud = {0}' } " />

< Etiqueta Texto = " {Binding Path = SizeRequest.Minimum,
StringFormat = 'Mínimo = {0}' } " />

< Etiqueta Texto = " {Binding Path = ElementBounds,
StringFormat = 'Límites = {0}' } " />
</ StackLayout >
</ StackLayout >
</ ScrollView >
</ Cuadrícula >
</ Página de contenido >
</ MasterDetailPage.Detail >
</ MasterDetailPage >
```

A continuación, puede establecer diversas combinaciones de VerticalOptions y HorizontalOptions sobre el Label y BoxView y ver cómo afectan los argumentos y valores de retorno de la GetSize-Solicitud método y los argumentos de la Diseño método (que se refleja en el Límites propiedad):



los VerticalOptions configuración de la Etiqueta y BoxView no tienen ningún efecto, excepto cuando el Explorar la bandera es cierto. los HorizontalOptions ajustes de posición de los elementos de la izquierda, centro o derecha.

Usted puede notar un par de rarezas: En primer lugar, la OpenStackLayout no recibe una llamada a su GetSizeRequest método. Es por esto que los tres primeros artículos en la mitad inferior de la pantalla son todos cero. Esta GetSizeRequest llamada normalmente provienen de la Cuadrícula, que es su padre. sin embargo, el Cuadrícula tiene un tamaño en función del tamaño de la pantalla, y el Cuadrícula contiene dos filas de igual tamaño. los OpenStackLayout tiene su VerticalOptions y HorizontalOptions propiedades ajustado a LayoutOptions.Fill, por lo que tendrá un tamaño que se basa en el Cuadrícula y no su contenido.

Si desea investigar más este comportamiento, tendrá que cambiar el VerticalOptions o HorizontalOptions propiedades de la OpenStackLayout en el marcado en el Detalle página. En ese caso, el Cuadrícula llamará a la GetSizeRequest método de OpenStackLayout -y OpenStackLayout luego hace GetSizeRequest las llamadas a Etiqueta y BoxView -porque necesita saber la Abierto-StackLayout tamaño a la posición él.

Ambos Abierto y OpenBoxView recibir llamadas a su GetSizeRequest métodos con restricciones de altura de Double.PositiveInfinity, pero el Etiqueta muestra algunas inconsistencias entre las plataformas.

En las diferentes plataformas de Windows, se desprende de los valores mostrados que la anchura de la restricción Etiqueta no es igual a la anchura de diseño de la StackLayout. Pero una exploración más profunda revela que la GetSizeRequest método es llamado más de una vez-la primera vez con el ancho de la disposición, y luego con el ancho deseado de la Etiqueta.

el androide Etiqueta devuelve la restricción de anchura que su anchura requerida, lo que significa que la horizontalOptions el establecimiento de la Etiqueta no tiene efecto en su posición horizontal. Esta diferencia en el

aplicación Android desaparece cuando el texto ocupa una sola línea.

Derivado de diseño <Ver>

Ahora estamos armados con el conocimiento suficiente para crear nuestras propias clases de diseño.

La mayoría de los métodos públicos y protegidos involucrados en el diseño se definen por la no genérica `Laico-fuera clase`. Los Disposición <T> clase se deriva de Diseño y limita el tipo genérico para Ver y sus derivados. Disposición <T> define una sola propiedad pública denominada Niños de tipo `IList<T>` y un par de métodos protegidos describe en breve.

Una clase de diseño personalizado casi siempre se deriva de Disposición <Ver>. Si desea restringir a los niños a ciertos tipos, puede derivar de Disposición <label> o Disposición <BoxView>, pero que no es común. (Usted verá un ejemplo hacia el final de este capítulo).

Una clase de diseño personalizado sólo tiene dos responsabilidades:

- Anular `OnSizeRequest` llamar `GetSizeRequest` en todos los niños del diseño. Devolver un tamaño requerido para el diseño en sí.
- Anular `layoutChildren` llamar `Diseño` en todos los niños del diseño.

Ambos métodos utilizan típicamente para cada o para para enumerar todos los niños en la costumbre de diseño Niños colección.

Es particularmente importante para la clase de diseño para llamar `Diseño` en cada niño. De lo contrario, el niño nunca consigue un tamaño o la posición correcta y no será visible.

Sin embargo, la enumeración de los niños en el `OnSizeRequest` y `layoutChildren` anulaciones deben omitir cualquier niño cuya `Es visible` propiedad se establece en falso. Estos niños no serán visibles de todos modos, pero si no se salte deliberadamente esos niños, lo más probable es que su diseño de clase dejará espacio para estos niños invisibles, y eso no es correcto comportamiento.

Como hemos visto, no se garantiza que la `OnSizeRequest` TRANSFERENCIA se llama. El método no necesita ser llamado si el tamaño del diseño se rige por su padre en lugar de sus hijos. El método definitivamente *será* ser llamado si una o ambas de las limitaciones son infinitos, o si la clase tiene la disposición de los valores no predeterminados `VerticalOptions` o `HorizontalOptions`. De lo contrario, una llamada a `OnSizeRequest` no se garantiza y no se debe confiar en él.

También ha visto que la `OnSizeRequest` llamada podría haber argumentos conjunto de restricciones a duplicó con `ble.PositiveInfinity`. Sin embargo, `OnSizeRequest` no puede devolver un tamaño solicitado con dimensiones infinitas. A veces hay una tentación de poner en práctica `OnSizeRequest` de una manera muy simple como esto:

```
// Esto es muy mal código!
protegido anular SizeRequest OnSizeRequest (doble widthConstraint, doble heightConstraint)
```

```
{
    return new SizeRequest ( nuevo tamaño (WidthConstraint, heightConstraint));
}
```

No lo haga! Si tu Disposición <Ver> derivado no puede hacer frente a las limitaciones infinitas, por alguna razón, y verá un ejemplo más adelante en este capítulo, a continuación, lanzar una excepción que indica que.

Muy a menudo, el layoutChildren anulación también requerirá conocer el tamaño de los niños. los layoutChildren método también se puede llamar GetSizeRequest en todos los niños antes de llamar Diseño. Es posible almacenar en caché el tamaño de los niños obtenidos en el OnSizeRequest anular para evitar más tarde GetSizeRequest llama en el layoutChildren anula, pero la clase de diseño tendrá que saber cuando los tamaños necesitan ser obtenida de nuevo. Verá algunas pautas breve.

Un ejemplo sencillo

Una buena técnica para aprender cómo escribir diseños personalizados es duplicar la funcionalidad de un diseño existente, pero simplificarlo un poco.

los VerticalStack clase se describe a continuación está destinado a imitar una StackLayout con un orientación ción ajuste de Vertical. los VerticalStack clase, por lo tanto no tiene una Orientación propiedad, y para simplificar las cosas, VerticalStack no tiene una Espaciado propiedad, ya sea. Además, VerticalStack no reconoce la expande bandera en el HorizontalOptions y Vertical- opciones la configuración de sus hijos. Haciendo caso omiso de la expande bandera simplifica la lógica de apilamiento *enormemente*.

VerticalStack por lo tanto, define sólo dos miembros: anulaciones de la OnSizeRequest y Laico-outChildren métodos. Típicamente, ambos métodos enumeran a través de la Niños Propiedad Definido por Disposición <T>, y en general los dos métodos hacen llamadas a la GetSizeRequest de los niños. Cualquier niño con una Es visible propiedad establecida en falso deben ser omitidos.

los OnSizeRequest SOBREPASO VerticalStack llamadas GetSizeRequest en cada niño con una restricción de anchura igual a la widthConstraint argumento para la anulación y una altura igual a restricción Double.PositiveInfinity. Esto limita la anchura del niño a la anchura de la Vertical-

Apilar, pero permite que cada niño sea tan alto como quiera. Esa es la característica fundamental de una pila vertical:

```
clase pública VerticalStack : Diseño < Ver >
{
    protegido anular SizeRequest OnSizeRequest ( doble widthConstraint,
                                                doble heightConstraint)
    {
        tamaño reqSize = nuevo tamaño ();
        tamaño minSize = nuevo tamaño ();

        // enumerar a través de todos los niños.
        para cada ( Ver niño en Niños )
        {
            // Saltar a los niños invisibles.
            Si (! Child.isVisible)
                continuar ;
        }
    }
}
```

```

// Obtener el tamaño requerido por el niño.
SizeRequest childSizeRequest = child.GetSizeRequest (widthConstraint,
    Double.PositiveInfinity);

// Encuentra la anchura máxima y acumular la altura.
reqSize.Width = Math.Max (reqSize.Width, childSizeRequest.Request.Width);
reqSize.Height += childSizeRequest.Request.Height;

// Hacer lo mismo para la solicitud de tamaño mínimo.
minSize.Width = Math.Max (minSize.Width, childSizeRequest.Minimum.Width);
minSize.Height += childSizeRequest.Minimum.Height;
}

return new SizeRequest (reqSize, minSize);
}
...
}

```

los para cada bucle sobre la Niños colección acumula el tamaño de los niños por separado para el Solicitud y Mínimo propiedades de la SizeRequest objeto devuelto por el niño. Estas acumulaciones implican dos tamaño valores, nombrados reqSize y minSize. Debido a que esta es una *vertical* pila, la

reqSize.Width y minSize.Width valores se establecen en el *máximo* de los anchos de niño, mientras que la reqSize.Height y minSize.Height valores se establecen en el *suma* de las alturas niño.

Es posible que el widthConstraint argumento para OnSizeRequest es Double.PositiveInfinite, en cuyo caso los argumentos de la GetSizeRequest llamada del niño son a la vez infinito. (Por ejemplo, el VerticalStack podría ser un hijo de una StackLayout con una orientación horizontal.) En general, el cuerpo de la OnSizeRequest no tiene que preocuparse por situaciones como que debido a la SizeRequest valor devuelto por GetSizeRequest Nunca contiene valores infinitos.

El segundo método, en un diseño de una costumbre de anulación layoutChildren -es muestra a continuación. Esto es generalmente denominada como consecuencia de una llamada a los padres de Diseño método.

los anchura y altura argumentos a layoutChildren indicar el tamaño del área del diseño disponible para sus hijos. Ambos valores son finitos. Si un argumento a OnSizeRequest era infinita, la correspondiente argumento para layoutChildren será la anchura o la altura de regresar de la OnSizeRequest. De lo contrario, depende de la HorizontalOptions y VerticalOptions ajustes. por Llenar, el argumento de layoutChildren es el mismo que el argumento correspondiente a En-SizeRequest. De lo contrario, es la anchura o la altura requerida de regresar de la OnSizeRequest.

layoutChildren también tiene X y y argumentos que reflejan la Relleno propiedad establecido en la disposición. Por ejemplo, si el relleno a la izquierda es 20 y el acolchado superior es 50, entonces X es 20 y y es de 50. Estos por lo general indican una posición de partida para los niños de la disposición:

```

pública clase VerticalStack : Diseño < Ver >
{
    ...
    protegido override void layoutChildren (doble X, doble Y, doble anchura, doble altura)
    {
        // enumerar a través de todos los niños.
    }
}

```

```

para cada ( Ver niño en Niños)
{
    // Saltar a los niños invisibles.

    Si (! Child.isVisible)
        continuar;

    // Obtener el tamaño requerido por el niño.

    SizeRequest childSizeRequest = child.GetSizeRequest (ancho, Doble .PositiveInfinity);

    // activación de la posición y el tamaño del niño.

    doble Xchild = x;
    doble yChild = y;
    doble childWidth = childSizeRequest.Request.Width;
    doble childHeight = childSizeRequest.Request.Height;

    // Ajustar la posición y el tamaño basado en HorizontalOptions.

    cambiar (Child.HorizontalOptions.Alignment)

    {
        caso LayoutAlignment .Comienzo:
            descanso;

        caso LayoutAlignment .Centrar:
            Xchild += (anchura - childWidth) / 2;
            descanso;

        caso LayoutAlignment .Fin:
            Xchild += (anchura - childWidth);
            descanso;

        caso LayoutAlignment .Llenar:
            childWidth = ancho;
            descanso;
    }

    // Disposición del niño.

    child.Layout (nuevo Rectángulo (Xchild, yChild, childWidth, childHeight));

    // Obtener la posición vertical del próximo hijo.

    y += childHeight;
}
}

```

Esta es una pila vertical, por lo que layoutChildren tiene que posicionar verticalmente cada niño basada en la altura requerida del niño. Si el niño tiene una `HorizontalOptions` ajuste de `Llenar`, a continuación, la anchura de cada niño es la misma que la anchura de la `VerticalStack` (menos el relleno). De lo contrario, la anchura del niño es su anchura solicitada, y la pila debe posicionar ese niño dentro de su propio ancho.

Para realizar estos cálculos, layoutChildren llama GetSizeRequest sobre sus hijos de nuevo, pero esta vez con la anchura real y argumentos a la altura layoutChildren en lugar de los argumentos de restricción utilizados en OnSizeRequest. A continuación, llama Diseño en cada niño, los altura argumento de la

Rectángulo constructor es siempre la altura del niño. los anchura argumento podría ser o bien el

anchura del niño o de la anchura de la **VerticalStack** pasó a la **layoutChildren** anular, dependiendo de la **HorizontalOptions** establecer en el niño. Observe que cada niño se coloca X unidades de la izquierda de la **VerticalStack**, y el primer niño se coloca y unidades de la parte superior de la **VerticalStack**. Ese y variable se aumenta entonces en la parte inferior del bucle en base a la altura del niño. Eso crea la pila.

los **VerticalStack** clase es parte de la **VerticalStackDemo** programa, que contiene una página de inicio que se desplaza a dos páginas para probarlo. Por supuesto, se puede añadir más páginas de prueba (que es algo que debe hacer para cualquier Disposición <Ver> clases que se desarrollan).

Las dos páginas de prueba se crean instancias en la página principal:

```
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    xmlns: sys = "clr-espacio de nombres: System; montaje = mscorelib "
    xmlns: locales = "clr-espacio de nombres: VerticalStackDemo; montaje = VerticalStackDemo "
    x: Class = "VerticalStackDemo.VerticalStackDemoHomePage "
    Título = "VerticalStack demostración ">

    < Vista de la lista itemSelected = " OnListViewItemSelected " >
        < ListView.ItemsSource >
            < x: Array Tipo = "x: Tipo de página" >
                < locales: LayoutOptionsTestPage />
                < locales: ScrollTestPage />
            </ x: Array >
        </ ListView.ItemsSource >

        < ListView.ItemTemplate >
            < DataTemplate >
                < TextCell Texto = "{Título} Encuadernación " />
            </ DataTemplate >
        </ ListView.ItemTemplate >

    </ Vista de la lista >
</ Pagina de contenido >
```

El archivo de código subyacente se desplaza a la página seleccionada:

```
público clase parcial VerticalStackDemoHomePage : Pagina de contenido
{
    público VerticalStackDemoHomePage ()
    {
        InitializeComponent ();
    }

    vacío asincrónico OnListViewItemSelected ( objeto remitente, SelectedItemChangedEventArgs args )
    {
        (( Vista de la lista ) Remitente) .SelectedItem = nulo ;

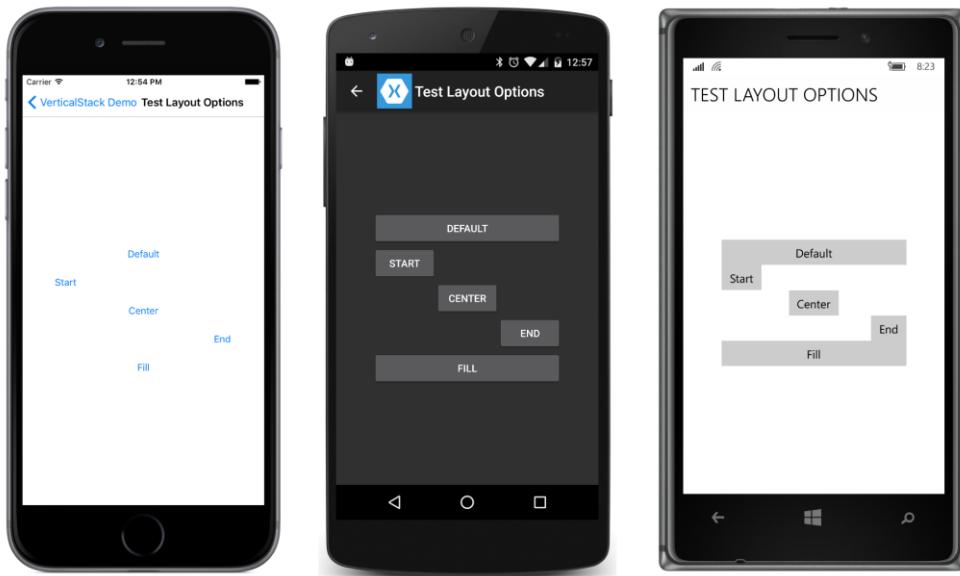
        Si (Args.SelectedItem = nulo )
        {
            Página page = ( Página ) Args.SelectedItem;
            esperar Navigation.PushAsync (página);
        }
    }
}
```

```
    }  
}
```

El primero de los usos páginas de prueba VerticalStack para mostrar cinco Botón elementos con diferentes HorizontalOptions ajustes. los VerticalStack sí se da una VerticalOptions Ajuste que debe colocarlo en el centro de la página:

```
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "  
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "  
    xmlns: locales = "clr-espacio de nombres: VerticalStackDemo; montaje = VerticalStackDemo "  
    x: Class = "VerticalStackDemo.LayoutOptionsTestPage "  
    Título = " Opciones de diseño de prueba " >  
  
< locales: VerticalStack Relleno = " 50, 0 "  
    VerticalOptions = " Centrar " >  
    < Botón Texto = " Defecto " />  
  
    < Botón Texto = " comienzo "   
        HorizontalOptions = " comienzo " />  
  
    < Botón Texto = " Centro "   
        HorizontalOptions = " Centro " />  
  
    < Botón Texto = " Fin "   
        HorizontalOptions = " Fin " />  
  
    < Botón Texto = " Llenar "   
        HorizontalOptions = " Llenar " />  
    </ locales: VerticalStack >  
</ Pagina de contenido >
```

Efectivamente, la lógica de los diversos HorizontalOptions configuración de los hijos de verticalmente calStack parece funcionar:



Obviamente, la plataforma móvil de Windows 10 se beneficiarán de una separación entre los botones!

Si se quita la VerticalOptions el establecimiento de la VerticalStack, el VerticalStack será *no* recibir una llamada en absoluto a su OnSizeRequest anular. No hay necesidad de ello. Los argumentos a Diseño-Niños reflejará todo el tamaño de la página menos el Relleno, y la página no necesita saber cuánto espacio de la VerticalStack requiere.

El segundo programa de pruebas pone el VerticalStack en un ScrollView:

```
< Página de contenido xmlns = "http://xamarin.com/schemas/2014/forms"
    xmlns: x = "http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns: locales = "clr-espacio de nombres: VerticalStackDemo; montaje = VerticalStackDemo "
    x: Class = "VerticalStackDemo.ScrollTestPage"
    Título = "Desplazamiento de prueba" >

< ScrollView >
    < locales: VerticalStack x: Nombre = "apilar" />
</ ScrollView >

</ Página de contenido >
```

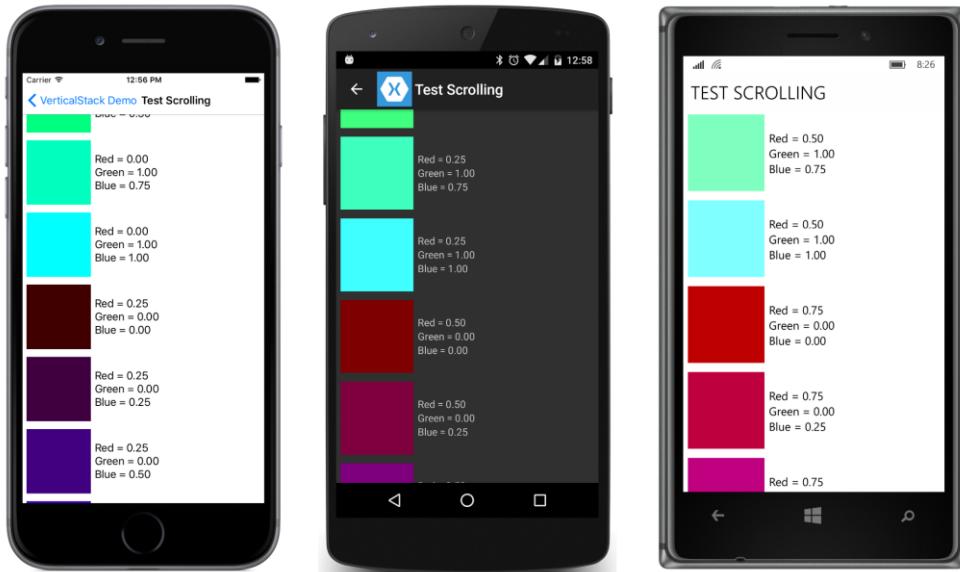
El archivo de código subyacente llena el VerticalStack con 125 casos de un habitual StackLayout, cada uno que contiene una BoxView, y otro VerticalStack con tres Etiqueta elementos:

```
público clase parcial ScrollTestPage : Página de contenido
{
    público ScrollTestPage ()
    {
        InitializeComponent ();

        para ( doble r = 0; r <= 1,0; r += 0,25)
            para ( doble g = 0; g <= 1,0; g += 0,25)
                para ( doble b = 0; b <= 1,0; b += 0,25)
```

```
{  
    stack.Children.Add ( nuevo StackLayout  
    {  
        Orientación = StackOrientation .Horizontal,  
        Relleno = 6,  
        Los niños =  
        {  
            nuevo BoxView  
            {  
                color = Color .FromRgb (r, g, b),  
                WidthRequest = 100,  
                HeightRequest = 100  
            },  
  
            nuevo VerticalStack  
            {  
                VerticalOptions = LayoutOptions .Centrar,  
                Los niños =  
                {  
                    nuevo Etiqueta {Texto = "Red =" + R.ToString ("F2")},  
                    nuevo Etiqueta {Texto = "= verdes" + G.ToString ("F2")},  
                    nuevo Etiqueta {Texto = "Azul =" + B.ToString ("F2")}  
                }  
            }  
        }  
    }  
});  
}  
}  
}
```

los VerticalStack es un hijo de una ScrollView con una orientación de desplazamiento vertical, por lo que recibe una OnSizeRequest llamar con una altura de Double.PositiveInfinity. los VerticalStack responde con una altura que abarca todos sus hijos. los ScrollView utiliza esa altura junto con su propia altura (que se basa en el tamaño de la pantalla) para desplazarse a su contenido:



posicionamiento vertical y horizontal simplifica

Hacia el final de la `layoutChildren` SOBREPASO `VerticalStack` es un cambiar declaración que ayuda en el posicionamiento de cada niño en posición horizontal basada en la del niño `HorizontalOptions` valor de la propiedad. Aquí está todo ese nuevo método:

```

pùblico class VerticalStack : Diseño < Ver >
{
    ...
    protegido override void layoutChildren ( doble X, doble Y, doble anchura, doble altura )
    {
        // enumerar a travéz de todos los niños.
        para cada ( Ver niñ en Niños )
        {
            // Saltar a los niños invisibles.
            Si ( ! Child.isVisible )
                continuar ;

            // Obtener el tamaño requerido por el niñ.
            SizeRequest childSizeRequest = child.GetSizeRequest ( ancho, Doble .PositiveInfinity );

            // activación de la posición y el tamaño del niñ.
            doble Xchild = x;
            doble yChild = y;
            doble childWidth = childSizeRequest.Request.Width;
            doble childHeight = childSizeRequest.Request.Height;

            // Ajustar la posición y el tamaño basado en HorizontalOptions.
            cambiar ( Child.HorizontalOptions.Alignment )
        }
    }
}

```

```

caso LayoutAlignment .Comienzo:
descanso ;

caso LayoutAlignment .Centrar:
Xchild += (anchura - childWidth) / 2;
descanso ;

caso LayoutAlignment .Fin:
Xchild += (anchura - childWidth);
descanso ;

caso LayoutAlignment .Llenar:
childWidth = ancho;
descanso ;
}

// Disposición del niño.

child.Layout (nuevo Rectángulo (Xchild, yChild, childWidth, childHeight));

// Obtener la posición vertical del próximo hijo.

y += childHeight;
}

}

```

La colocación de un niño dentro de un rectángulo en función de su HorizontalOptions y VerticalOptions
la configuración es algo que aparece con bastante frecuencia al escribir diseños. Por esa razón, el Laico-
cabo <T> clase incluye un método estático público que lo hace por usted:

pùblico hoyo estatico LayoutChildIntoBoundingRegion (VisualElement niñ o, Rectángulo región)

Puede volver a escribir la `layoutChildren` método a utilizar este método de ayuda de este modo:

```
protegido override void layoutChildren ( doble X, doble Y, doble anchura, doble altura)
{
    // enumerar a través de todos los niños.

    para cada ( Ver niño en Niños)
    {
        // Saltar a los niños invisibles.

        Si (I Child.isVisible)
            continuar ;

        // Obtener el tamaño requerido por el niño.

        SizeRequest childSizeRequest = child.GetSizeRequest (ancho, Doble .PositiveInfinity);

        doble childHeight = childSizeRequest.Request.Height;

        // Disposición del niño.

        LayoutChildIntoBoundingRegion (niño, nuevo Rectángulo (X, y, ancho, childHeight));

        // Calcular la posición vertical próximo hijo.

        y + = childHeight;
    }
}
```

Eso es una simplificación considerable! Pero a medida que esta llamada se utiliza en otras clases de diseño en este capítulo, tenga en cuenta que es equivalente a hacer una llamada a la del niño Diseño método.

Tenga en cuenta que el rectángulo que se pasa a LayoutChildIntoBoundingRegion abarca toda la zona en la que el niño puede residir. En este caso, el ancho argumento de la Rectángulo constructor es el ancho argumento pasado a layoutChildren, que es la anchura de la VerticalLayout sí mismo. Pero el alto argumento de la Rectángulo constructor es la altura del niño en particular requiere, que está disponible en GetSizeRequest.

A menos que el niño tiene por defecto HorizontalOptions y VerticalOptions ajustes de Llenar, el LayoutChildIntoBoundingRegion método en sí tiene que llamar GetSizeRequest en el niño mediante el Anchura y Altura propiedades de ese Rectángulo valor. Esa es la única manera que sabe cómo colocar al niño en el ámbito previsto en ese Rectángulo se pasa a la llamada al método.

Esto significa que cuando se utiliza el LayoutChildIntoBoundingRegion método, el VerticalLayout fuera clase muy bien podría llamar GetSizeRequest tres veces en cada niño en cada ciclo de diseño.

Además, así como VerticalLayout llamadas GetSizeRequest en sus hijos varias veces, y en ocasiones con diferentes argumentos, el padre de VerticalLayout podría llamar GetSizeRequest sobre el VerticalLayout más de una vez con diferentes argumentos, que luego se traduce en más OnSizeRequest llamadas.

Las llamadas a GetSizeRequest no debería tener ningún efecto secundario. Las llamadas no dan lugar a ninguna otra propiedad que se establecen, y deben simplemente recuperar la información en base a determinadas limitaciones de ancho y alto. GetSizeRequest por lo tanto puede ser llamado más libremente que Diseño, que en realidad afecta a cómo el elemento está dimensionado y posicionado.

Pero no llame GetSizeRequest si no es necesario. Una llamada a GetSizeRequest no se requiere para un elemento que se mostrará en la pantalla. Solamente Diseño es requerido.

En sus propias clases de diseño, es mejor para manejar OnSizeRequest llama "a ciegas", sin tratar de averiguar donde la llamada está vieniendo, o por qué los argumentos son lo que son, o lo que significa obtener varias llamadas con diferentes argumentos.

Sin embargo, es posible para su clase de diseño para almacenar en caché el resultado de la OnSizeRequest llama para que pueda agilizar las llamadas posteriores. Pero hacer esto correctamente requiere conocer sobre el proceso de *invalidación*.

Invalidación

Supongamos que ha reunido a algunos diseños y puntos de vista sobre una página, y por alguna razón el archivo de código subyacente (o tal vez un disparador o comportamiento) cambia el texto de una Botón, o tal vez sólo un tamaño de fuente o atributo. Ese cambio podría afectar el tamaño del botón, lo que potencialmente podría tener un efecto dominó de los cambios en el diseño a través del resto de la página.

El proceso por el cual un cambio en un elemento de la página desencadena una nueva disposición se conoce como *invalidación*. Cuando algo en la página no es válido, es significa que ya no tiene un tamaño o posición correcta. Se requiere un nuevo ciclo de diseño.

El proceso de invalidación comienza con un método virtual protegido definido por VisualElement:

```
protegido virtual void InvalidateMeasure ()
```

Este método está protegida. No se puede invalidar un elemento de código externo. Los elementos deben invalidar a sí mismos, por lo general cuando una propiedad de los cambios de los elementos. Esto ocurre comúnmente en las implementaciones de propiedades enlazables. Cada vez que hay un cambio en una de las propiedades enlazables del elemento que pudiera dar lugar a un nuevo tamaño del elemento, el manejador de propiedad cambiado llama usualmente EnvalidateMeasure.

los InvalidateMeasure método dispara un evento de manera que cualquier objeto externo al elemento podría ser informado cuando el elemento ya no tiene un tamaño correcto:

```
evento público Controlador de eventos MeasureInvalidated;
```

El padre del elemento generalmente se encarga de esta MeasureInvalidated evento. Sin embargo, el elemento no hace nada más allá de disparar este evento. No cambia su propio tamaño de diseño. Esa es la responsabilidad del parent del elemento. Sin embargo, cualquier futura llamada a GetSizeRequest reflejará el nuevo tamaño.

VisualElement sí define 28 propiedades públicas, pero sólo unos pocos de ellos gatillo llamadas a invalidateMeasure y una cocción posterior de la MeasureInvalidated evento. Estas propiedades son:

- Es visible
- WidthRequest y MinimumWidthRequest
- HeightRequest y MinimumHeightRequest

Estas son las únicas propiedades que VisualElement define que causan un cambio en el tamaño del diseño del elemento.

VisualElement define algunas propiedades que pueden provocar un cambio en la *apariencia* del elemento, pero no un cambio en el tamaño del diseño. Estos son BackgroundColor, IsEnabled, IsFocused, y Opacidad. Los cambios en estas propiedades no causan las llamadas a InvalidateMeasure.

En adición, VisualElement define ocho transformar las propiedades que cambian el tamaño de un elemento de las prestaciones, no cambian el tamaño del elemento como se percibe en la disposición. Estos son AnchorX, Anchory, Rotación, RotationX, rotationY, Escala, TranslationX, y TranslationY.

los Conductas, estilo, y disparadores propiedades podrían *indirectamente* afectar tamaño de diseño, pero los cambios en estas propiedades (o las colecciones que mantienen estas propiedades) no lo hacen ellos mismos causa invalida measure ser llamado. Además, los cambios a la InputTransparent, navegación, y Rfuentes propiedades no afectan tamaño del diseño.

Y luego están las cinco propiedades que se establecen por una llamada a Diseño. Estos son Límites, X, Y,

Anchura, y Altura. Estas propiedades deben definitivamente no-y no-causar una llamada a invalida dateMeasure.

los Ver clase añade tres propiedades más a los definidos por VisualElement. Los GestureRecognizers propiedad no afecta tamaño de diseño, pero los cambios en las dos propiedades siguientes causan una llamada a InvalidateMeasure:

- HorizontalOptions
- VerticalOptions

Las clases que se derivan de Ver También hacer llamadas a InvalidateMeasure siempre que una propiedad cambios que podrían causar un cambio en el tamaño del elemento. Por ejemplo, Etiqueta llamadas InvalidateMeasure:

Ure siempre que cualquiera de las siguientes propiedades cambian:

- Texto y FormattedText
- Familia tipográfica, Tamaño de fuente, y FontAttributes
- LineBreakMode

Etiqueta hace *no* llamada InvalidateMeasure cuando el Color de texto cambios de propiedad. Eso afecta a la apariencia del texto pero no su tamaño. Etiqueta también lo hace *no* llamada InvalidateMeasure cuando el horizontalTextAlignment y VerticalTextAlignment propiedades cambian. Estas propiedades rigen la alineación del texto dentro del tamaño total de la Etiqueta, pero no afectan el tamaño de la Label sí mismo.

Los Diseño clase se basa en la infraestructura de la invalidación de varias maneras cruciales. Primero, Diseño define un método similar al InvalidateMeasure llamado InvalidateLayout:

```
protegido virtual void InvalidateLayout ()
```

UN Diseño la clase derivada debe llamar InvalidateLayout cada vez que se realiza un cambio que afecta a cómo las posiciones de clase disposición y el tamaño de sus hijos.

Los Diseño misma clase llamadas InvalidateLayout cada vez que se añade o se retira de un niño su Contenido propiedad (en el caso de ContentView, Marco, y ScrollView) o su Niños colección (en el caso de Disposición <Ver> derivados).

Si lo haces *no* quiere que su clase de diseño para llamar InvalidateLayout cuando se añade o elimina un niño, puede anular la ShouldInvalidateOnChildAdded y ShouldInvalidateOnChildRemoved métodos y simplemente retorno falso en lugar de cierto. Su clase se puede implementar un proceso personalizado cuando se añaden o eliminan los niños. Los Disposición <T> clase anula los métodos virtuales denominadas OnChildAdded y OnChildRemoved definida por la Elemento clase, pero su clase en lugar debe anular el OnAdded y OnRemoved métodos para el procesamiento personalizado.

Además, el Diseño clase establece un controlador para el MeasureInvalidated evento en todos los niños

añadido a su Contenido la propiedad o Niños colección, y se desprende el controlador cuando se retira el niño. los Página clase hace algo similar. Ambos Página y Diseño clases exponen reemplazable OnChildMeasureInvalidated métodos si desea ser notificado cuando se disparan estos eventos.

Estas MeasureInvalidated manejadores de eventos son realmente la parte crucial del proceso, porque cada elemento en el árbol visual que tiene hijos es alertado cada vez que uno de sus hijos cambia de tamaño. Así es como un cambio en el tamaño de un elemento muy profundo en el árbol visual puede causar cambios que ondulan el árbol.

los Diseño clase, sin embargo, los intentos de limitar el impacto de un cambio en el tamaño de un niño en el diseño total de la página. Si el diseño particular, está limitada en tamaño, entonces un cambio en el tamaño de un niño no tiene por qué afectar algo mayor que este diseño en el árbol visual.

En la mayoría de los casos, un cambio en el tamaño de un diseño afecta la forma en la disposición organiza sus hijos. Por esta razón, cualquier cambio en el tamaño de un diseño precipitará un ciclo de diseño para el diseño. La disposición obtendrá llamadas a su OnSizeRequested y layoutChildren métodos.

Sin embargo, lo contrario no siempre es cierto. La forma en que una disposición organiza sus hijos podrían afectar el tamaño de la disposición, o tal vez no. Lo más obvio es el tamaño de la disposición se *no* verá afectada por la forma en la disposición organiza sus hijos si el tamaño de la disposición está totalmente restringido.

Esta diferencia se hace importante cuando el diseño define sus propias propiedades, tales como la Espaciado y Orientación propiedades definidas por StackLayout. Cuando una propiedad cambia de valor, el diseño debe invalidar sí para causar un nuevo ciclo de diseño que se produzca. Pudiera la disposición InvalidateMedida o InvalidateLayout?

En la mayoría de los casos, el diseño debe llamar InvalidateLayout. Esto garantiza que el diseño recibe una llamada a su layoutChildren Método incluso si el diseño está totalmente restringido en tamaño. Si las llamadas de diseño InvalidateMeasure, a continuación, un nuevo pase diseño se generará únicamente si el diseño no está restringido totalmente en tamaño. Si el diseño se ve limitada en tamaño, a continuación, una llamada a InvalidateMeasure no hará nada.

Algunas reglas para los diseños de codificación

De la discusión anterior, se puede formular varias reglas para su propio Disposición <Ver> derivados:

Regla 1: Si la clase de diseño define las propiedades tales como Espaciado o Orientación, estas propiedades deben ser respaldados por las propiedades enlazables. En la mayoría de los casos, los manipuladores de propiedad-cambiado de estas propiedades enlazables deben llamar InvalidateLayout. Vocación InvalidateMeasure debería limitarse a los casos en que un cambio de propiedad afecta al tamaño de la disposición y no la forma en que organiza sus hijos, sino un ejemplo de la vida real es difícil de imaginar.

Regla 2: La clase de diseño puede definir las propiedades enlazables adjunto para sus hijos similares a la Fila, Columna, RowSpan, y ColumnSpan propiedades definidas por Cuadrícula. Como se sabe, estas propiedades están definidas por la clase de diseño, sino que están destinados a ser fijado sobre los hijos de la disposición. En este caso, la clase de diseño debe anular el OnAdded método para añadir una PropertyChanged manejador a cada niño de la disposición, y anulación OnRemoved para eliminar ese controlador. los PropertyChanged

manejador debe comprobar si la propiedad que se cambió en el niño es una de las propiedades enlazables adjuntas que su clase ha definido, y si es así, su diseño debe responder normalmente llamando `invalidateLayout`.

Regla 3: Si se desea implementar una memoria caché (o retener otra información) para reducir al mínimo el procesamiento repetitivo de las llamadas a la `GetSizeRequest` métodos de los niños del diseño, entonces también deberían anular el `InvalidateLayout` método para ser notificado cuando se añaden a los niños a o se eliminan de la disposición, y el `OnChildMeasureInvalidated` Método para ser notificado cuando uno de los niños del diseño de los cambios de tamaño. En ambos casos, la clase de diseño debe responder en la limpieza de caché o descartar que la información retenida.

Es posible que el diseño también para borrar la caché o descartar información retenida cuando la disposición recibe una llamada a su `InvalidateMeasure` método. Sin embargo, en general, la memoria caché es un diccionario basado en tamaños pasado a la `OnSizeRequest` y `layoutChildren` anulan, por lo que los tamaños serán diferentes modos.

Todas estas técnicas se demostrarán en las páginas que siguen.

Un diseño con propiedades

los `StackLayout` es ciertamente útil, pero es sólo una única fila o columna de los niños. Si desea que varias filas y columnas, se puede utilizar el Cuadrícula, pero la aplicación debe establecer explícitamente el número de filas y columnas, y que requiere tener una buena idea del tamaño de los niños.

Un diseño más útil para dar cabida a un número indefinido de los niños comenzaría el posicionamiento de los niños en una fila horizontal, al igual que una `StackLayout`, pero luego ir a una segunda fila si es necesario, y una tercera, sin embargo continua para muchas filas son necesarios. Si se espera que el número de filas que se supere la altura de la pantalla, a continuación, la disposición se podría hacer un niño de un `ScrollView`.

Esta es la idea detrás `WrapLayout`. Se organiza sus hijos en columnas horizontalmente a través de la pantalla hasta que se llega al borde, momento en el que se envuelve la pantalla de los niños posteriores a la siguiente fila, y así sucesivamente.

Pero vamos a hacer un poco más versátil: Vamos a darle una `Orientación` propiedad como `StackLayout`.

Esto permite el uso de un programa de `WrapLayout` para especificar que comienzan por la organización de sus hijos en filas abajo de la pantalla, y luego debe ir a una segunda columna si es necesario. Con esta orientación alternativa, la `WrapLayout` podría ser horizontal desplazado.

También vamos a dar `WrapLayout` dos propiedades, el nombre `ColumnSpacing` y `Distancia entre filas`, al igual que

Cuadrícula.

los `WrapLayout` tiene el potencial de ser bastante complejo algorítmicamente si realmente permite a los niños de una variedad de diferentes tamaños. La primera fila podría tener cuatro hijos, tres niños en la segunda fila, y así sucesivamente.

En lugar de eso hacer una simple suposición de que todos los niños tienen el mismo tamaño, o más precisamente,

que la misma cantidad de espacio se asigna para cada niño en función del tamaño máximo de los niños. Esto a veces se llama una *tamaño de la celda*, y WrapLayout calculará un tamaño de celda que es lo suficientemente grande para todos los niños. Los niños más pequeños que el tamaño de la celda se pueden colocar dentro de esa celda en función de su horizontalOptions y VerticalOptions ajustes.

WrapLayout es lo suficientemente útil como para justificar su inclusión en el **Xamarin.FormsBook.Toolkit** biblioteca. La siguiente enumeración contiene las dos opciones de orientación con descripciones prolifas, pero sin ambigüedades:

```
espacio de nombres Xamarin.FormsBook.Toolkit
{
    public enum WrapOrientation
    {
        HorizontalThenVertical,
        VerticalThenHorizontal
    }
}
```

WrapLayout define tres propiedades respaldados por propiedades enlazables. El controlador de la propiedad cambió de cada propiedad enlazable simplemente llama InvalidateLayout para desencadenar un nuevo pase de diseño en el diseño:

```
espacio de nombres Xamarin.FormsBook.Toolkit
{
    clase pública WrapLayout : Diseño < Ver >
    {

        ...
        sólo lectura estática pública BindableProperty OrientationProperty =
        BindableProperty .Crear(
            "Orientación",
            tipo de ( WrapOrientation ),
            tipo de ( WrapLayout ),
            WrapOrientation .HorizontalThenVertical,
            PropertyChanged: (enlazable, oldValue, newValue) =>
            {
                (( WrapLayout ) Enlazable) .InvalidateLayout ();
            });
        ...

        sólo lectura estática pública BindableProperty ColumnSpacingProperty =
        BindableProperty .Crear(
            "ColumnSpacing",
            tipo de ( doble ),
            tipo de ( WrapLayout ),
            6.0,
            PropertyChanged: (enlazable, OldValue, nuevovalue) =>
            {
                (( WrapLayout ) Enlazable) .InvalidateLayout ();
            });
        ...

        sólo lectura estática pública BindableProperty RowSpacingProperty =
        BindableProperty .Crear(
            "Distancia entre filas",
            tipo de ( doble ),
```

```

    tipo de ( WrapLayout ),
    6.0,
    PropertyChanged: (enlazable, OldValue, nuevovalor) =>
    {
        (( WrapLayout ) Enlazable).InvalidateLayout ();
    });

    público WrapOrientation Orientación
    {
        conjunto {EstablecerValor (OrientationProperty, valor );}
        obtener {regreso ( WrapOrientation ) GetValue (OrientationProperty);}
    }

    pública doble ColumnSpacing
    {
        conjunto {EstablecerValor (ColumnSpacingProperty, valor );}
        obtener {regreso ( doble ) GetValue (ColumnSpacingProperty);}
    }

    pública doble Distancia entre filas
    {
        conjunto {EstablecerValor (RowSpacingProperty, valor );}
        obtener {regreso ( doble ) GetValue (RowSpacingProperty);}
    }
    ...
}

}

```

WrapLayout también define una estructura privada para almacenar información acerca de una colección particular de los niños. Los tamaño de celda propiedad es el tamaño máximo de todos los niños, pero ajustado al tamaño de la disposición. Los filas y cols propiedades son el número de filas y columnas.

```

espacio de nombres Xamarin.FormsBook.Toolkit
{
    clase pública WrapLayout : Diseño < Ver >
    {
        struct LayoutInfo
        {
            público LayoutInfo (En t visibleChildCount, tamaño tamaño de celda, En t files, En t cols): este ()
            {
                VisibleChildCount = visibleChildCount;
                Tamaño de celda = tamaño de celda;
                Filas = filas;
                Cols = cols;
            }

            public int VisibleChildCount {conjunto privado ; obtener ;}

            público tamaño tamaño de celda conjunto privado ; obtener ;}

            public int files conjunto privado ; obtener ;}

            public int cols {conjunto privado ; obtener ;}
        }
    }
}
```

```

Diccionario < tamaño , LayoutInfo > = LayoutInfoCache nuevo Diccionario < tamaño , LayoutInfo > 0;
...
}
}

```

Nótese también la definición de una Diccionario para almacenar múltiples LayoutInfo valores. los tamaño clave es cualquiera de los argumentos de restricciones a la OnSizeRequest anular, o la anchura y altura argumentos a la layoutChildren anular.

Si el WrapLayout está en una restringida ScrollView (que normalmente será el caso), entonces uno de los argumentos de restricción será infinito, pero que no será el caso para el anchura y altura argumentos a LayoutChildren. En ese caso, habrá dos entradas del diccionario.

Si a continuación, gire el teléfono hacia los lados, WrapLayout tendrá otra OnSizeRequest llamar con una restricción infinita, y otro layoutChildren llamada. Eso es más de dos entradas del diccionario. Pero entonces si a su vez el teléfono a modo vertical, no hay más cálculos tienen por qué producirse porque la caché ya cuenta con ese caso.

Aquí está el GetLayoutInfo método en el WrapLayout que calcula las propiedades de la Diseño-información estructura basada en un tamaño particular. Observe que el método comienza comprobando si una calculada LayoutInfo valor ya está disponible en la caché. Al final de GetLayoutInfo método, el nuevo LayoutInfo valor se almacena en la memoria caché:

```

espacio de nombres Xamarin.FormsBook.Toolkit
{
    clase pública WrapLayout : Diseño < Ver >
    {
        ...
        LayoutInfo GetLayoutInfo ( doble anchura, doble altura)
        {
            tamaño size = nuevo tamaño (Anchura, altura);

            // Comprobar si la información está disponible en caché.
            Si (LayoutInfoCache.ContainsKey (tamaño))
            {
                regreso layoutInfoCache [tamaño];
            }

            En t visibleChildCount = 0;
            tamaño maxChildSize = nuevo tamaño ();
            En t filas = 0;
            En t cols = 0;
            LayoutInfo layoutInfo = nuevo LayoutInfo ();

            // enumerar a través de todos los niños.
            para cada ( Ver niño en Niños)
            {
                // Saltar niños invisibles.
                Si (! Child.IsVisible)
                    continuar ;

```

```
// Cuenta los hijos visibles.  
visibleChildCount ++;  
  
// Obtener el tamaño requerido por el niño.  
SizeRequest childSizeRequest = child.GetSizeRequest ( Doble.PositiveInfinity,  
Doble.PositiveInfinity);  
  
// acumular el tamaño máximo del niño.  
maxChildSize.Width =  
    Mates.MAX (maxChildSize.Width, childSizeRequest.Request.Width);  
  
maxChildSize.Height =  
    Mates.MAX (maxChildSize.Height, childSizeRequest.Request.Height);  
}  
  
Si (visibleChildCount == 0)  
{  
    // Calcular el número de filas y columnas.  
    Si (Orientación == WrapOrientation.HorizontalThenVertical)  
    {  
        Si (Doble.IsPositiveInfinity (ancho))  
        {  
            cols = visibleChildCount;  
            filas = 1;  
        }  
        más  
        {  
            cols = (Ent) ((Ancho + ColumnSpacing) /  
                         (MaxChildSize.Width + ColumnSpacing));  
            cols = Mates.MAX (1, cols);  
            filas = (visibleChildCount + cols - 1) / cols;  
        }  
    }  
    más // WrapOrientation.VerticalThenHorizontal  
    {  
        Si (Doble.IsPositiveInfinity (altura))  
        {  
            filas = visibleChildCount;  
            cols = 1;  
        }  
        más  
        {  
            filas = (Ent) ((Altura + RowSpacing) /  
                           (MaxChildSize.Height + RowSpacing));  
            filas = Mates.MAX (1, filas);  
            cols = (visibleChildCount + filas - 1) / filas;  
        }  
    }  
}  
  
// Ahora maximizar el tamaño de la celda basándose en el tamaño del diseño.  
tamaño = tamaño de celda nuevo tamaño 0;  
  
Si (Doble.IsPositiveInfinity (ancho))  
{
```

```

        cellSize.Width = maxChildSize.Width;
    }

    más

    {
        cellSize.Width = (anchura - ColumnSpacing * (cols - 1)) / cols;
    }
}

Si ( Doble .IsPositiveInfinity (altura))
{
    cellSize.Height = maxChildSize.Height;
}
más
{
    cellSize.Height = (altura - RowSpacing * (filas - 1)) / filas;
}

layoutInfo = nuevo LayoutInfo (visibleChildCount, tamaño de celda, filas cols);
}

layoutInfoCache.Add (tamaño, layoutInfo);

regreso layoutInfo;

}

...
}
}

```

La lógica de GetLayoutInfo se divide en tres secciones principales:

La primera sección es una para cada bucle que enumera a través de todos los niños, las llamadas `GetSizeRequest` con una anchura infinita y la altura, y determina el tamaño máximo de niño.

Las secciones segunda y tercera se ejecutan sólo si hay al menos un hijo visibles. La segunda sección ha de procesamiento diferente, basado en la Orientación propiedad y calcula el número de filas y columnas. Por lo general será el caso de que una WrapPanel con el valor por defecto Orientación ajuste (horizontalThenVertical) será un niño de un verticales ScrollView, en cuyo caso el heightConstraint argumento de la OnSizeRequest override será infinito. También podría darse el caso de que la widthConstraint argumento para OnSizeRequest (y GetLayoutInfo) También es infinito, lo que da lugar a todos los niños que se muestran en una sola fila. Pero eso sería inusual.

La tercera sección a continuación, calcula un tamaño de celda para los niños en base a las dimensiones de la WrapLayout. Por un Orientación de HorizontalThenVertical, este tamaño celular suele ser un poco más ancho que el tamaño máximo de los niños, pero podría ser menor si el WrapLayout no es lo suficientemente amplio como para que el niño más ancho o la altura suficiente para que el niño más alto.

La memoria caché debe ser destruido por completo cuando la disposición recibe llamadas a `InvalidateLayout` (lo que podría resultar cuando se añaden a los niños a o retirados de la colección, o cuando una de las propiedades de `WrapLayout` valor cambios) o para `OnChildMeasureInvalidate`. Esto es simplemente una cuestión de desmontar el diccionario:

espacio de nombres Xamarin.FormsBook.Toolkit

```

clase pública WrapLayout : Diseño < Ver >
{
    ...
    protegido override void InvalidateLayout ()
    {
        base .InvalidateLayout ();

        // Descartar toda la información de diseño para los niños añadidos o eliminados.
        layoutInfoCache.Clear ();
    }

    protegido override void OnChildMeasureInvalidated ()
    {
        base .OnChildMeasureInvalidated ();

        // Descartar toda la información de diseño para el tamaño del niño cambiado.
        layoutInfoCache.Clear ();
    }
}
}

```

Por último, estamos listos para mirar a los dos métodos requeridos. los OnSizeRequest simplemente pide anular GetLayoutInfo y construye una SizeRequest el valor de la información devuelta junto con el Distancia entre filas y ColumnSpacing propiedades:

```

espacio de nombres Xamarin.FormsBook.Toolkit
{
    clase pública WrapLayout : Diseño < Ver >
    {
        ...
        protegido anular SizeRequest OnSizeRequest ( doble widthConstraint,
                                                    doble heightConstraint)
        {
            LayoutInfo layoutInfo = GetLayoutInfo (widthConstraint, heightConstraint);

            Si (LayoutInfo.VisibleChildCount == 0)
            {
                return new SizeRequest ();
            }

            tamaño TotalSize = nuevo tamaño (LayoutInfo.CellSize.Width * + layoutInfo.Cols
                                              ColumnSpacing * (layoutInfo.Cols - 1),
                                              layoutInfo.CellSize.Height * + layoutInfo.Rows
                                              RowSpacing * (layoutInfo.Rows - 1));

            return new SizeRequest (tamaño total);
        }
        ...
    }
}

```

los layoutChildren anulación comienza con una llamada a GetLayoutInfo y luego enumera todos los niños con el tamaño y la posición de ellos dentro de la celda de cada niño. Esta lógica también requiere un procesamiento separado

basado en el Orientación propiedad:

```

espacio de nombres Xamarin.FormsBook.Toolkit
{
    clase pública WrapLayout : Diseño < Ver >
    {
        ...
        protected void override layoutChildren ( doble X, doble Y, doble anchura, doble altura)
        {
            LayoutInfo layoutInfo = GetLayoutInfo (anchura, altura);

            Si (LayoutInfo.VisibleChildCount == 0)
                regreso ;

            doble Xchild = x;
            doble yChild = y;
            En t fila = 0;
            En t col = 0;

            para cada ( Ver niño en Niños)
            {
                Si (! Child.isVisible)
                    continuar ;

                LayoutChildIntoBoundingRegion (niño,
                    nuevo Rectángulo (nuevo Punto (Xchild, yChild), layoutInfo.CellSize));

                Si (Orientación == WrapOrientation.HorizontalThenVertical)
                {
                    Si (++ col == layoutInfo.Cols)
                    {
                        col = 0;
                        fila++;
                        Xchild = x;
                        yChild += RowSpacing + layoutInfo.CellSize.Height;
                    }
                    más
                    {
                        Xchild += ColumnSpacing + layoutInfo.CellSize.Width;
                    }
                }
                más // == Orientación WrapOrientation.VerticalThenHorizontal
                {
                    Si (++ fila == layoutInfo.Rows)
                    {
                        col++;
                        fila = 0;
                        Xchild += ColumnSpacing + layoutInfo.CellSize.Width;
                        yChild = y;
                    }
                    más
                    {
                        yChild += RowSpacing + layoutInfo.CellSize.Height;
                    }
                }
            }
        }
    }
}

```

Vamos a probarlo! El archivo XAML de la **PhotoWrap** programa contiene simplemente una WrapPanel con valores de propiedades por defecto en una ScrollView:

```
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    xmlns: kit de herramientas =
        "clr-espacio de nombres: Xamarin.FormsBook.Toolkit; montaje = Xamarin.FormsBook.Toolkit
    x: Class = " PhotoWrap.PhotoWrapPage " >

< ContentPage.Padding >
    < OnPlatform x: TypeArguments = " Espesor "
        iOS = " 0, 20, 0, 0 " />
</ ContentPage.Padding >

< ScrollView >
    < kit de herramientas: WrapLayout x: Nombre = " wrapLayout " />
</ ScrollView >

</ Pagina de contenido >
```

El archivo de código subyacente accede al archivo JSON que contiene la lista de fotos de archivo utilizados previamente en varios programas de ejemplo en este libro. El constructor crea una `Imagen` elemento para cada mapa de bits en la lista y lo añade a la `WrapLayout`:

```

público clase parcial PhotoWrapPage : Pagina de contenido

{
    [DataContract]
    clase ImageList

    {
        [DataMember (Nombre = "fotos")]
        público Lista< cuerdas > Fotos = nulo ;

    }
}

WebRequest solicitud;
estático int sólo lectura imageDimension = Dispositivo .OnPlatform (240, 240, 120);
estático cadena de sólo lectura urlSuffix =
    Cuerda .Formato( "? Width = {0} & height = {0} & mode = max" , ImageDimension);

público PhotoWrapPage ()
{
    InitializeComponent ();

    // lista de fotografías de archivo Obtener.
    Uri uri = nuevo Uri ( "Http://docs.xamarin.com/demo/stock.json" );
    request = WebRequest .Create (URI);
    request.BeginGetResponse (WebRequestCallback, nulo );

}

vacio WebRequestCallback ( IAsyncResult resultado)
{
    string resultado = resultado .Result .ToString ();
}

```

```

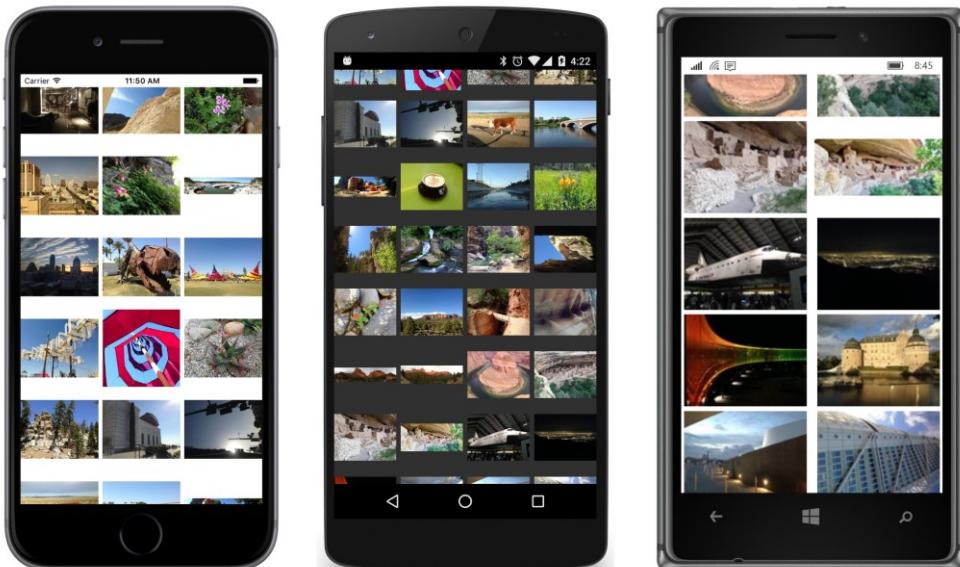
{
    tratar
    {
        Corriente flujo = request.EndGetResponse (resultado).GetResponseStream ();

        // deserializar el JSON en imageList.
        var jsonSerializer = nuevo DataContractJsonSerializer ( tipo de ( ImageList ));
        ImageList imageList = ( ImageList ) jsonSerializer.ReadObject (corriente);

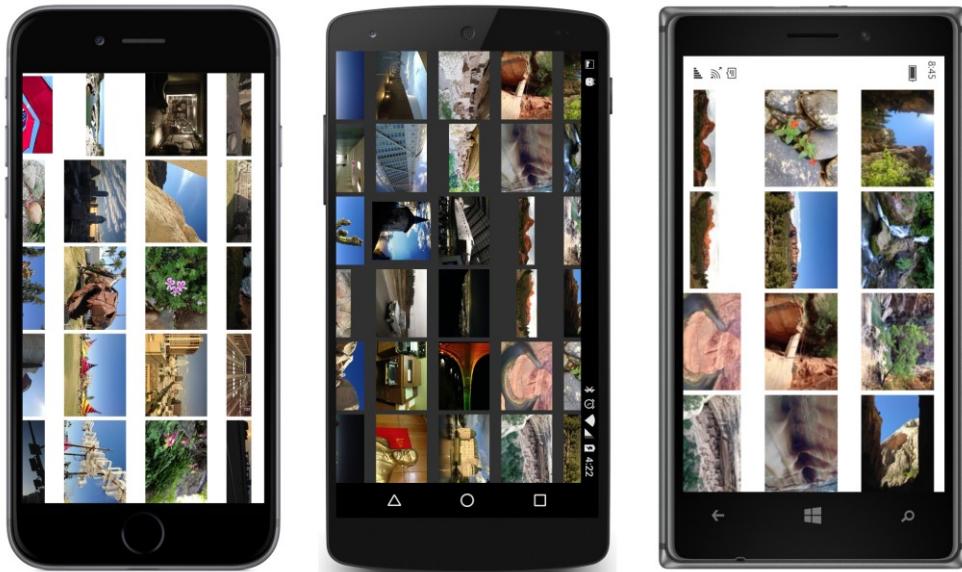
        Dispositivo .BeginInvokeOnMainThread (() =>
        {
            para cada ( cuerda ruta de archivo en imageList.Photos)
            {
                Imagen image = nuevo Imagen
                {
                    fuente = Fuente de imagen .FromUri ( nuevo Uri (Filepath + urlSuffix));
                };
                wrapLayout.Children.Add (imagen);
            }
        });
    }
    captura ( Excepción )
    {
    }
}
}

```

El número de columnas en cada fila depende del tamaño de los mapas de bits, el ancho de la pantalla, y el número de píxeles por unidad independiente del dispositivo:



A su vez los teléfonos de lado, y verá algo un poco diferente:



los ScrollView permite la disposición que se desplaza verticalmente. Si desea comprobar la diferente orientación de la WrapPanel, tendrá que cambiar la orientación de la ScrollView también:

```
<ScrollView Orientación = "Horizontal">
    <kit de herramientas:WrapLayout x: Nombre = "wrapLayout"
        Orientación = "VerticalThenHorizontal" />
</ScrollView>
```

Ahora la pantalla se desplaza horizontalmente:



los `Imagen` elementos se cargan los mapas de bits en el fondo, por lo que la `WrapLayout` clase obtendrá numerosas llamadas a su `Diseño` método ya que cada `Imagen` elemento obtiene un nuevo tamaño basándose en el mapa de bits cargado. En consecuencia, es posible que aparezca algún desplazamiento de las filas y columnas como los mapas de bits se están cargando.

No hay dimensiones sin restricciones permitidas

Hay momentos en que desea ver *todo* en la pantalla, tal vez en una matriz de filas y columnas de tamaño uniforme. Usted puede hacer algo como esto con una Cuadrícula con todas las definiciones de filas y columnas definidas con el asterisco para que sean todas del mismo tamaño. El único problema es que es probable que también desea que el número de filas y columnas que se basa en el número de niños, y optimizado para el mejor uso del espacio de la pantalla.

Vamos a escribir un diseño personalizado llamado `UniformGridLayout`. Me gusta `WrapLayout`, `UniformGridLayout` requiere `Orientación`, `Distancia entre filas`, y `ColumnSpacing` propiedades, así que vamos a eliminar algunas de las tareas realizadas en la redefinición de las propiedades derivando `UniformGridLayout` de `WrapLayout`.

Porque `UniformGridLayout` no tiene sentido con una dimensión sin restricciones, el `OnSize-` solicitud invalidar los controles de las limitaciones infinitas y plantea una excepción si se encuentra con una cosa así.

Para ayudar en la capacidad de las `UniformGridLayout` para optimizar el uso del espacio de la pantalla, vamos a darle una propiedad denominada `AspectRatio` de tipo `AspectRatio`. Esta propiedad indica la relación de aspecto esperado de los niños como una doble valor. El valor de 1,33, por ejemplo, indica una relación de aspecto de 4: 3, que es una anchura que es 33 por ciento mayor que la altura. Por defecto, sin embargo, `UniformGridLayout` fuera calculará una relación de aspecto promedio de sus hijos.

Esta `AspectRatio` estructura es similar a la `GridLength` estructura definida para la `Cuadrícula` clase, ya que permite una doble valor, así como una opción “Auto” para forzar `UniformGridLayout` calcular

que la relación de aspecto promedio:

```
espacio de nombres Xamarin.FormsBook.Toolkit
{
    [ TypeConverter ( tipo de ( AspectRatioTypeConverter ) )]
    público struct AspectRatio
    {
        público AspectRatio ( doble valor )
        {
            Si (Valor <0)
                arrojar nueva FormatException ( "AspectRatio valor debe ser mayor que 0" +
                    "O se establece en 0 para indicar Auto" );
            Valor = valor;
        }

        public static AspectRatio Auto
        {
            obtener
            {
                return new AspectRatio ();
            }
        }

        pública doble El valor { conjunto privado ; obtener ; }

        public bool IsAuto { obtener { regreso Valor == 0; } }

        público cadena de anulación Encadenar()
        {
            regreso Valor == 0? "Auto" : Value.ToString ();
        }
    }
}
```

La opción “Auto” se indica mediante una Valor propiedad de 0. Una aplicación que utiliza UniformGridLayout puede crear una tal AspectRatio valor con el constructor sin parámetros, o pasando un 0 al constructor definido, o mediante el uso de la estática Auto propiedad.

Estoy seguro de que le gustaría ser capaz de establecer una AspectRatio propiedad en XAML, por lo que la estructura se marca con una TypeConverter atributo. Los AspectRatioTypeConverter clase puede manejar una cadena con la palabra “Auto” o una doble:

```
espacio de nombres Xamarin.FormsBook.Toolkit
{
    clase pública AspectRatioTypeConverter : TypeConverter
    {
        público bool anulación CanConvertFrom ( Tipo tipo de fuente)
        {
            regreso sourceType == tipo de ( cuerda );
        }

        público objeto de anulación ConvertFrom ( CultureInfo cultura, objeto valor)
        {

```

```

cuerda str = valor como cuerda ;

Si ( Cuerda .IsNullOrEmptyOrWhiteSpace (str))
    return null ;

str = str.Trim ();
doble aspectValue;

Si ( Cuerda .Compare (str, "auto", StringComparison .OrdinalIgnoreCase) == 0)
    regreso AspectRatio .Auto;

Si ( Doble .TryParse (str, NumberStyles .Número,
    CultureInfo .InvariantCulture, fuera aspectValue))
    return new AspectRatio (AspectValue);

arrojar nueva FormatException ( "AspectRatio debe ser automático o numérico" );

}
}

```

Los `UniformGridLayout` clase se deriva de `WrapLayout` únicamente para heredar las tres propiedades enlazables que `WrapLayout` define. Para esas propiedades, `UniformGridLayout` agrega el `AspectRatio` propiedad:

```
espacio de nombres Xamarin.FormsBook.Toolkit

{
    clase pública UniformGridLayout : WrapLayout

    {
        sólo lectura estática pública BindableProperty AspectRatioProperty =
            BindableProperty .Crear(
                "AspectRatio",
                tipo de ( AspectRatio ),
                tipo de ( UniformGridLayout ),
                AspectRatio .Auto,
                PropertyChanged: (enlazable, OldValue, nuevousuario) =>
                {
                    (( UniformGridLayout ) Enlazable) .InvalidateLayout ();
                });
    }

    público AspectRatio AspectRatio
    {
        conjunto {EstablecerValor (AspectRatioProperty, valor); }
        obtener {regreso ( AspectRatio ) GetValue (AspectRatioProperty); }
    }
}

...
```

los `OnSizeRequest` anulación comienza comprobando si las restricciones son infinitos y lanzar una excepción si ese es el caso. De lo contrario, solicita toda el área a menos que no tiene hijos visibles:

```
espacio de nombres Xamarin.FormsBook.Toolkit  
{  
    clase pública UniformGridLayout : WrapLayout
```

```

{

...
protegido anular SizeRequest OnSizeRequest ( doble widthConstraint,
                                              doble heightConstraint
{
    Si ( Doble .IsInfinity (widthConstraint) || Doble .IsInfinity (heightConstraint)
        arrojar nueva InvalidOperationException (
            "UniformGridLayout no se puede utilizar con dimensiones sin restricciones." );

    // Sólo tienes que comprobar para ver si no hay niños visibles.

    En t childCount = 0;

    para cada ( Ver ver en Niños)
        childCount += view.isVisible? 1: 0;

    Si (ChildCount == 0)
        return new SizeRequest ();

    // Entonces solicitar todo el tamaño (noninfinite).
    return new SizeRequest ( nuevo tamaño (WidthConstraint, heightConstraint));
}

...
}
}

```

La parte difícil es la `layoutChildren` anular, y tiene tres secciones principales:

```
espacio de nombres Xamarin.FormsBook.Toolkit

{
    clase pública UniformGridLayout : WrapLayout
    {
        ...
        protegido override void layoutChildren ( doble X, doble Y, doble anchura, doble altura)
        {
            En t childCount = 0;

            para cada ( Ver ver en Niños)
                childCount + = view.isVisible? 1: 0;

            Si (ChildCount == 0)
                regreso ;

            doble childAspect = AspectRatio.Value;

            // Si AspectRatio es automático, calcular una relación de aspecto media
            Si (AspectRatio.IsAuto)
            {
                En t nonZeroChildCount = 0;
                doble accumAspectRatio = 0;

                para cada ( Ver ver en Niños)
                {
                    Si (View.isVisible)
                    {
                        SizeRequest sizeRequest = view.GetSizeRequest ( Doble .PositiveInfinity,
```

```
Doble .PositiveInfinity);  
  
    Si (SizeRequest.Request.Width > 0 && sizeRequest.Request.Height > 0)  
    {  
        nonZeroChildCount++;  
        accumAspectRatio += sizeRequest.Request.Width /  
                           sizeRequest.Request.Height;  
    }  
}  
}  
  
Si (NonZeroChildCount > 0)  
{  
    childAspect = accumAspectRatio / nonZeroChildCount;  
}  
más  
{  
    childAspect = 1;  
}  
}  
  
En t bestRowsCount = 0;  
En t bestColsCount = 0;  
doble bestUsage = 0;  
doble bestChildWidth = 0;  
doble bestChildHeight = 0;  
  
// test diversas posibilidades de que el número de columnas.  
para (En t colsCount = 1; colsCount <= childCount; colsCount++)  
{  
    // Buscar el número de filas para que muchas columnas.  
    En t rowsCount = (En t) Mates.Techo((doble) ChildCount / colsCount);  
  
    // Determinar si tenemos más filas o columnas de las que necesitamos.  
    Si ((RowsCount - 1) * colsCount >= childCount ||  
        rowsCount * (colsCount - 1) >= childCount)  
    {  
        continuar;  
    }  
  
    // Obtener la relación de aspecto de las células resultantes.  
    doble cellWidth = (ancho - ColumnSpacing * (colsCount - 1)) / colsCount;  
    doble cellHeight = (altura - RowSpacing * (rowsCount - 1)) / rowsCount;  
    doble cellAspect = cellWidth / cellHeight;  
    doble uso = 0;  
  
    // comparar con la relación de aspecto promedio del niño.  
    Si (CellAspect > childAspect)  
    {  
        uso = childAspect / cellAspect;  
    }  
    más  
{  
    uso = cellAspect / childAspect;
```

La primera sección calcula una relación de aspecto promedio de los niños si la opción "Auto" se ha especificado.

La segunda sección de bucle a través de diferentes combinaciones de filas y columnas y determina qué combinación resulta en el mejor uso del espacio disponible. El cálculo crucial es la siguiente:

```
Si (CellAspect> childAspect)
{
    uso = childAspect / cellAspect;
}
más
{
    uso = cellAspect / childAspect;
}
```

Por ejemplo, supongamos que la `childAspect` que se calcula basándose en el promedio de todos los niños es 1,5, y para una combinación particular de filas y columnas las `cellAspect` valor es 2. Un niño con una relación de aspecto de 1,5 ocupará sólo el 75 por ciento de una célula con una relación de aspecto de 2. Si el `cellAspect` es en cambio 0,75, entonces el niño ocupa sólo el 50 por ciento de esa célula.

La tercera sección da entonces a cada niño un tamaño y posición dentro de la cuadrícula. Esto requiere un procesamiento diferente en función de la Orientación propiedad.

Vamos a probarlo. Los **Mosaico de imágenes** archivo XAML llena la página (excepto para el relleno superior en el iPhone) con una `UniformGridLayout` con dos propiedades establecen:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    xmlns: kit de herramientas =
        "clr-espacio de nombres: Xamarin.FormsBook.Toolkit; montaje = Xamarin.FormsBook.Toolkit "
    x: Class = " PhotoGrid.PhotoGridPage " >

< ContentPage.Padding >
    < OnPlatform x: TypeArguments = " Espesor "
        iOS = " 0, 20, 0, 0 " />
</ ContentPage.Padding >

< kit de herramientas: UniformGridLayout x: Nombre = " uniformGridLayout "
    Orientación = " VerticalThenHorizontal "
    AspectRatio = " Auto " />

</ Página de contenido >
```

El archivo de código subyacente es prácticamente idéntica a la de **PhotoWrap**, y aquí está el resultado:



De nuevo, como la `Imagen` elementos se cargan los mapas de bits, es posible que vea algún desplazamiento de las filas y columnas.

Es divertido para ejecutar esto en el escritorio de Windows y cambiar el tamaño y la relación de aspecto de la ventana para ver cómo los mapas de bits se reordenan en filas y columnas. Esta es una buena manera también para comprobar si hay algunos errores en su código.

La superposición de los niños

¿puede una Disposición <Ver> llamada clase del Diseño Método de sus hijos para que los niños se superponen? Sí, pero que probablemente plantea otra pregunta en su mente: ¿Qué determina el orden en que los niños se vuelven? ¿Qué niños aparentemente se sientan en el primer plano y podrían parcial o totalmente oscuros otros niños que se muestra en el fondo?

En algunos entornos gráficos, programadores tienen acceso a un valor llamado *Z-index*. El nombre proviene de la visualización de un sistema de coordenadas tridimensional en una pantalla de ordenador en dos dimensiones. Los ejes X e Y definen la superficie horizontal de la pantalla, mientras que el eje Z es perpendicular a la pantalla. Los elementos visuales con un mayor índice Z parecen estar más cerca del espectador en el primer plano, y por lo tanto elementos podrían posiblemente oscuros con un menor índice Z en el fondo.

No hay una explícita índice Z en Xamarin.Forms. Se puede suponer que un índice Z está implícito en el orden en que la clase de diseño de la llama Diseño Método de sus hijos, pero este no es el caso. Una clase de diseño puede llamar al Diseño métodos en sus hijos en cualquier orden que deseé sin ningún cambio en la pantalla. Estas llamadas dan a cada niño un tamaño y posición relativa a su padre, pero los niños son *no* dictada en ese orden.

En cambio, los niños se prestan en su orden en la Niños colección. Los hijos anteriores de la colección se procesan primero, para que aparezcan en el fondo, lo que significa que los niños más adelante en

la colección parece estar en el primer plano y puede oscurecer los niños anteriores.

los Diseño clase define dos métodos que le permiten moverse a un niño al principio o al final de la Niños colección. Estos métodos son:

- LowerChild - mueve un niño al comienzo de la Niños colección, y visualmente a un segundo plano.
- RaiseChild - mueve un niño al final de la Niños colección, y visualmente al primer plano.

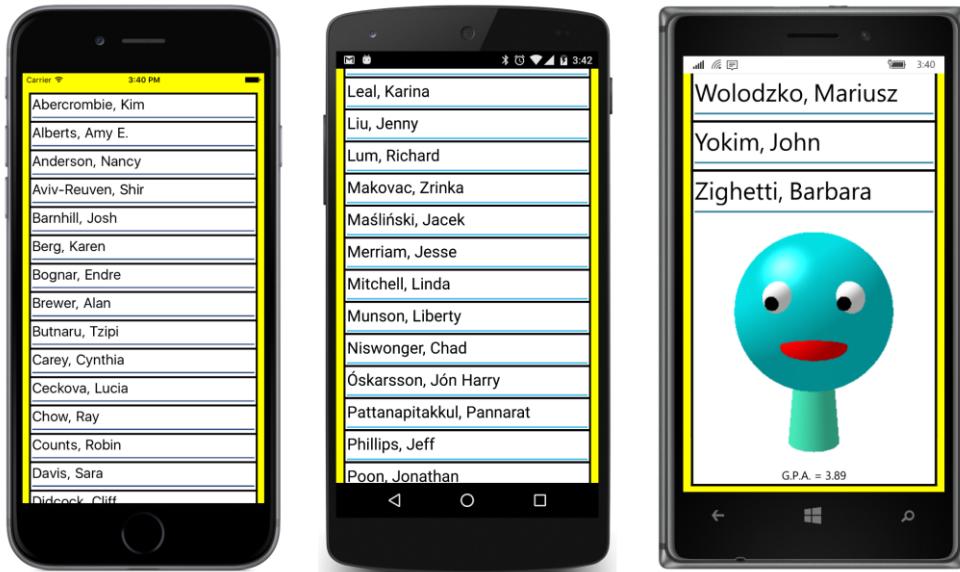
El niño ya debe ser una parte de la Niños recogida de estos métodos para trabajar. Estas llamadas resultan en una llamada a la protegida OnChildrenReordered método definido por VisualElement y un disparo del ChildrenReordered evento.

En el momento de escribir este capítulo, el LowerChild y RaiseChild métodos no funcionan en las diferentes plataformas de Windows. sin embargo, el Niños Propiedad Definido por Disposición <T> es de tipo IList <T>, lo que también puede mover los niños dentro y fuera de la colección con las llamadas a Añadir, Insertar, Re-movimiento, y RemoveAt. Independientemente de cómo lo hace, cualquier cambio en el contenido de la Niños resultados de la recogida en una llamada a LayoutInvalidated y un nuevo ciclo de diseño.

Estos problemas surgen cuando se quiere escribir una clase de diseño que se superpone a sus hijos, sino que también quieren la opción de llevar a un niño parcialmente oculta de su escondite, quizás con un toque. Para mover a un niño al primer plano visual, que necesita para manipular el Niños colección, pero también tendrá que asegurarse de que estas manipulaciones no interfieran con la prestación de los niños.

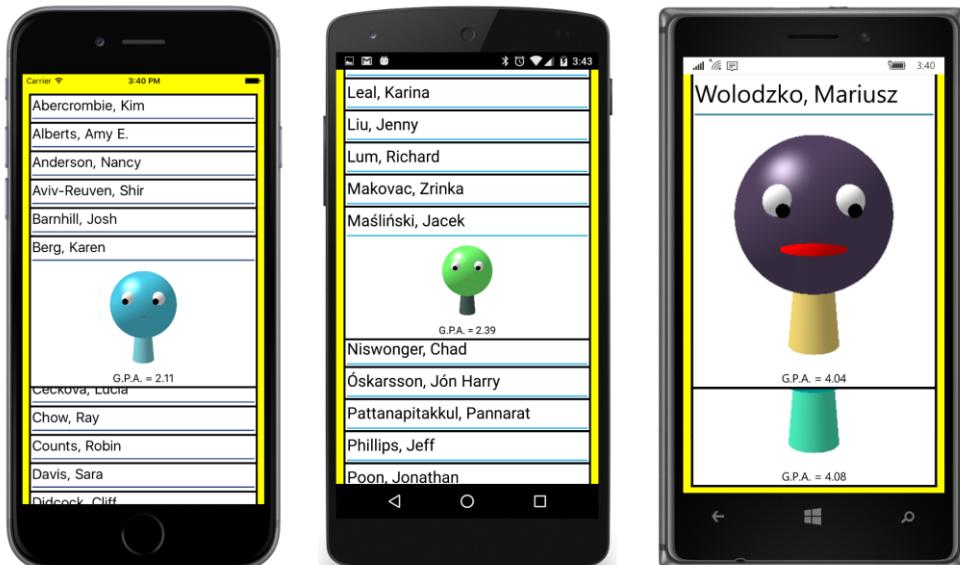
Vas a ver una posible solución en el OverlapLayout clase. Esta clase de diseño muestra sus hijos en una pila vertical u horizontal, pero se superponen. Cada niño se coloca ligeramente más baja (o a la derecha de) el niño anterior, especificada por una propiedad que OverlapLayout define llamada Compensar.

Aquí se llama programa de StudentCardFile que usa OverlapLayout en un ScrollView para mostrar a los estudiantes de la Escuela de Arte por el uso de una metáfora tarjeta de archivo:



Los estudiantes están clasificadas por apellido. La pantalla de iOS muestra la parte superior de la lista. La pantalla de Android se desplaza a algún lugar en el medio, y la pantalla de Windows Mobile 10 se desplaza hasta el final. El estudiante sólo visible por completo es el que al final de la Niños colección, con un apellido muy tarde en el alfabeto.

Para ver un estudiante, puede tocar la parte superior de la tarjeta de estudiante:



El niño es llevado al primer plano con llamadas a dos métodos que simulan una `RaiseChild` llamada:

```
overlapLayout.Children.Remove (tappedChild);
overlapLayout.Children.Add (tappedChild);
```

Ahora puede desplazarse por la lista como si fuera normal. Todos los niños están en el mismo orden de arriba a abajo. Puede causar ese niño a ser restaurado a su posición inicial en el Niños colección con otro toque en ese niño o tocando a otro niño.

Si se piensa en la lógica de VerticalStack anteriormente en este capítulo, se puede imaginar que puede haber un poco de un problema si sólo tiene que llamar RaiseChild sin hacer nada más. RaiseChild envía al niño al final de la Niños colección, por lo que normalmente quedaría pasado y aparece en la parte inferior de la lista. Necesitamos una manera de reordenar la Niños colección manteniendo constante el orden de procesamiento.

La solución que OverlapLayout utiliza es una propiedad enlazable adjunto que puede establecer en cada niño por la aplicación. Esta propiedad se llama RenderOrder, y verá cómo funciona en breve.

Aquí es cómo definir una propiedad enlazable adjunta en una clase de diseño. Es un poco diferente de una propiedad enlazable regulares:

```
espacio de nombres Xamarin.FormsBook.Toolkit
{
    clase pública OverlapLayout : Diseño < Ver >
    {
        ...
        // Asociado propiedad enlazable.
        sólo lectura estática pública BindableProperty RenderOrderProperty =
            BindableProperty.CreateAttached ("RenderOrder",
                tipo de ( Ent ),
                tipo de ( OverlapLayout ),
                0);

        // métodos de ayuda para la propiedad enlazable adjunto.
        pública static void SetRenderOrder ( bindableObject enlazable, Ent orden )
        {
            bindable.SetValue ( RenderOrderProperty, orden );
        }

        int pública static GetRenderOrder ( bindableObject enlazable )
        {
            regreso ( Ent ) Bindable.GetValue ( RenderOrderProperty );
        }
        ...
    }
}
```

La definición del public static campo de sólo lectura es similar a la definición de una propiedad que puede vincularse normal, excepto que se utiliza la estática Bindable.CreateAttached método, que define al menos el nombre de texto de la propiedad, el tipo de la propiedad, del tipo de la clase que define la propiedad, y un valor predeterminado.

Sin embargo, a diferencia de una propiedad enlazable regular, lo hace *no* definir una propiedad de C#. En su lugar, se definen dos métodos estáticos para establecer y obtener la propiedad. Estos dos métodos llamados estáticos de ayuda

SetRenderOrder y GetRenderOrder -son no es estrictamente necesario. Cualquier código que utiliza la propiedad enlazable adjunta sólo tiene que llamar Valor ajustado y GetValue En cambio, como los cuerpos de los métodos demuestran. Pero ellos están acostumbrados.

Como verá, código o marcado usando OverlapLayout que diferencia a este RenderOrder la propiedad de cada uno de los niños del diseño. Los StudentCardFile la muestra se verá en breve establece la propiedad cuando los niños son primero creadas y nunca cambia. Sin embargo, en el caso general, las propiedades enlazables adjuntas establecidos en los niños pueden cambiar, en cuyo caso se requiere otro pase diseño.

Por esta razón, los diseños que implementan propiedades enlazables adjuntas deben anular el OnAdded y OnRemoved métodos de colocar (y soltar) un controlador para el PropertyChanged evento en cada niño en el Niños colección de la disposición. Este controlador comprueba los cambios en la propiedad enlazable adjunta e invalida la disposición, si el valor de la propiedad ha cambiado:

```
espacio de nombres Xamarin.FormsBook.Toolkit
{
    clase pública OverlapLayout : Diseño <Ver>
    {
        ...
        // monitor PropertyChanged eventos para los elementos de la colección de los niños.

        protegido override void OnAdded ( Ver ver )
        {
            base .OnAdded ( vista );
            view.PropertyChanged += OnChildPropertyChanged;
        }

        protegido override void OnRemoved ( Ver ver )
        {
            base .OnRemoved ( vista );
            view.PropertyChanged -= OnChildPropertyChanged;
        }

        vacío OnChildPropertyChanged ( objeto remitente, PropertyChangedEventArgs args )
        {
            Si (Args.PropertyName == "RenderOrder")
            {
                InvalidateLayout ();
            }
        }
        ...
    }
}
```

En lugar de hacer referencia de forma explícita el nombre de texto de la propiedad en el PropertyChanged manejador (y posiblemente falta de ortografía es), se puede hacer referencia, alternativamente, la Nombre de la propiedad propiedad de la RenderOrderProperty inmueble objeto enlazable.

OverlapLayout también define dos propiedades enlazables regulares. Los Orientación propiedad se basa en el vigente StackOrientation enumeración (debido a que el diseño es muy similar a una pila) y Compensar indica la diferencia entre cada niño sucesiva:

```
espacio de nombres Xamarin.FormsBook.Toolkit

{
    clase pública OverlapLayout : Diseño < Ver >
    {
        sólo lectura estática pública BindableProperty OrientationProperty =
            BindableProperty .Crear(
                "Orientación",
                tipo de ( StackOrientation ),
                tipo de ( OverlapLayout ),
                StackOrientation .Vertical,
                PropertyChanged: (enlazable, oldValue, newValue) =>
                {
                    (( OverlapLayout ) Enlazable) .InvalidateLayout ();
                });
    }

    sólo lectura estática pública BindableProperty OffsetProperty =
        BindableProperty .Crear(
            "Compensar",
            tipo de ( doble ),
            tipo de ( OverlapLayout ),
            20.0,
            PropertyChanged: (enlazable, OldValue, nuevovalor) =>
            {
                (( OverlapLayout ) Enlazable) .InvalidateLayout ();
            });
    ...

    público StackOrientation Orientación
    {
        conjunto {EstablecerValor (OrientationProperty, valor); }
        obtener { regreso ( StackOrientation ) GetValue (OrientationProperty); }
    }

    pública doble Compensar
    {
        conjunto {EstablecerValor (OffsetProperty, valor); }
        obtener { regreso ( doble ) GetValue (OffsetProperty); }
    }
    ...
}

}
```

Los dos sustituciones de método requeridas son bastante simples en comparación con algunas de las otras clases de diseño en este capítulo. OnSizeRequest simplemente determina el tamaño máximo de los niños y calcula un tamaño solicitado en base al tamaño de un niño, porque inicialmente sólo un niño es totalmente visible- más el producto de la Compensar el valor y el número de niños menos uno:

Si nosotros no tenemos que preocuparnos de llevar a los niños escondidos en el primer plano, el layoutChildren método sería posicionar cada niño sucesiva incrementando X o Y (dependiendo de la orientación) por

Compensar unidades. En cambio, el método calcula una childOffset valor para cada niño multiplicando el

Compensar propiedad por parte del RenderOrder propiedad:

```
espacio de nombres Xamarin.FormsBook.Toolkit
{
    clase pública OverlapLayout : Diseño < Ver >
    {
        ...
        protected void override layoutChildren ( doble X, doble Y, doble anchura, doble altura)
        {
            para cada ( Ver niño en Niños)
            {
                Si (! Child.isVisible)
                    continuar ;

                SizeRequest childSizeRequest = child.GetSizeRequest (anchura, altura);
                doble childOffset = Offset * GetRenderOrder (niño);

                Si (Orientación == StackOrientation.Vertical)
                {
                    LayoutChildIntoBoundingRegion (niño,
                        nuevo Rectángulo (X, y + childOffset,
                            anchura, childSizeRequest.Request.Height));
                }
                más // == Orientación StackOrientation.Horizontal
                {
                    LayoutChildIntoBoundingRegion (niño,
                        nuevo Rectángulo (X + childOffset, y,
                            childSizeRequest.Request.Width, altura));
                }
            }
        }
    }
}
```

La declaración que realiza la multiplicación de la Compensar y el RenderOrder propiedad es

```
doble childOffset = Offset * GetRenderOrder (niño);
```

Puede hacer lo mismo sin la estática GetRenderOrder mediante el uso de la propiedad GetValue:

```
doble childOffset = Offset * ( En t ) Child.GetValue (RenderOrderProperty);
```

Pero el GetRenderOrder método es definitivamente más fácil.

los StudentCardFile programa define una página con una OverlapLayout en un ScrollView:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    xmlns: kit de herramientas =
        "clr-espacio de nombres: Xamarin.FormsBook.Toolkit; montaje = Xamarin.FormsBook.Toolkit "
    x: Class = " StudentCardFile.StudentCardFilePage "
    Color de fondo = " Amarillo " >

< ContentPage.Padding >
    < OnPlatform x: TypeArguments = " Espesor "
```

```
iOS = "0, 20, 0, 0" />
</ContentPage.Padding >

<ScrollView >
<kit de herramientas: OverlapLayout x: Nombre = "overlapLayout" 
Relleno = "10" />
</ScrollView >
</Pagina de contenido >
```

El archivo de código subyacente instancia el SchoolViewModel y utiliza el PropertyChanged evento para determinar cuando el Cuerpo de estudiantes propiedad es válida. Los estudiantes están ordenados primero por el apellido. Luego, para cada Estudiante objeto, el código crea una StudentView (que verá en breve) y asigna el

Estudiante oponerse a la vista de BindingContext:

```
público clase parcial StudentCardFilePage : Pagina de contenido
{
    ...
    público StudentCardFilePage ()
    {
        InitializeComponent ();

        // Establecer una plataforma específica Offset en el OverlapLayout.
        overlapLayout.Offset = 2 * Dispositivo .GetNamedSize ( NamedSize .Grande, tipo de ( Etiqueta ));

        SchoolViewModel viewmodel = nuevo SchoolViewModel ();

        viewModel.PropertyChanged += (remitente, args) =>
        {
            Si (Args.PropertyName == "Cuerpo de estudiantes" )
            {
                // Clasificar los estudiantes por apellido.
                var estudiantes =
                    viewModel.StudentBody.Students.OrderBy (estudiante => student.LastName);

                Dispositivo .BeginInvokeOnMainThread () =>
                {
                    En t index = 0;

                    // Recorrer los estudiantes.
                    para cada ( Estudiante estudiante en estudiantes )
                    {
                        // Crear un StudentView para cada uno.
                        StudentView studentView = nuevo StudentView
                        {
                            BindingContext = estudiante
                        };

                        // Establecer la Orden adjunta propiedad enlazable.
                        OverlapLayout .SetRenderOrder (studentView, índice ++);

                        // Adjuntar un controlador Tap gesto.
                        TapGestureRecognizer tapGesture = nuevo TapGestureRecognizer ();
                        tapGesture.Tapped += OnStudentViewTapped;
                        studentView.GestureRecognizers.Add (tapGesture);
                    };
                };
            };
        };
    };
}
```

```
// Añadir a la OverlapLayout.  
overlapLayout.Children.Add (studentView);  
    }  
});  
}  
};  
}  
...  
}
```

los RenderOrder propiedad se establece simplemente que los valores secuenciales:

```
OverlapLayout .SetRenderOrder (studentView, índice ++);
```

No parece mucho, pero es crucial para mantener el orden de representación de los estudiantes cuando el Niños se altera colección.

Los Niños colección se altera en el aprovechado entrenador de animales. Tenga en cuenta que el código tiene que manejar tres casos diferentes (pero relacionadas): Un toque en una tarjeta de estudiante requiere que la tarjeta se colocará en primer plano con la manipulación de **la Niños colección**, equivalente a una llamada a `RaiseChild` - excepto si la tarjeta de estudiante ya está en el primer plano, en cuyo caso se necesita la tarjeta para volver a poner donde estaba. Si una tarjeta ya está en el primer plano cuando se pulsa otra tarjeta, a continuación, la primera tarjeta se debe mover hacia atrás y la segunda tarjeta colocará en primer plano:

```
pùblico clase parcial StudentCardFilePage : Pagina de contenido
{
    Ver exposedChild = nulo ;

    ...
    void OnStudentViewTapped ( objeto remitente, EventArgs args)
    {
        Ver tappedChild = (Ver )remitente;
        bool retractOnly = tappedChild == exposedChild;

        // Repliegue el niñ o expuesto.
        Si (ExposedChild!= nulo )
        {
            overlapLayout.Children.Remove (exposedChild);
            overlapLayout.Children.Insert (
                OverlapLayout .GetRenderOrder (exposedChild), exposedChild);
            exposedChild = nulo ;
        }

        // Exponer a un nuevo hijo.
        Si (! RetractOnly)
        {
            // Levante niñ o.
            overlapLayout.Children.Remove (tappedChild);
            overlapLayout.Children.Add (tappedChild);

            exposedChild = tappedChild;
        }
    }
}
```

los StudentView clase se deriva de ContentView y está destinado a parecerse a una tarjeta de índice. Las fronteras son delgadas BoxView elementos, y otro BoxView dibuja una línea horizontal bajo el nombre en la parte superior de la tarjeta:

```
<ContentView xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns:x = " http://schemas.microsoft.com/winfx/2009/xaml "
    x:Class = "StudentCardFile.StudentView "
    Color de fondo = "Blanco">

<ContentView.Resources>
    <ResourceDictionary>
        <x: Doble x: Key = "espesor">3</x: Doble>

        <Estilo Tipo de objetivo = "Etiqueta">
            <Setter Propiedad = "Color de texto" Valor = "Negro" />
        </Estilo>

        <Estilo Tipo de objetivo = "BoxView">
            <Setter Propiedad = "Color" Valor = "Negro" />
        </Estilo>
    </ResourceDictionary>
</ContentView.Resources>

<Cuadricula>
    <BoxView VerticalOptions = "comienzo"
        HeightRequest = "(Espesor) StaticResource" />

    <BoxView VerticalOptions = "Fin"
        HeightRequest = "(Espesor) StaticResource" />

    <BoxView HorizontalOptions = "comienzo"
        WidthRequest = "(Espesor) StaticResource" />

    <BoxView HorizontalOptions = "Fin"
        WidthRequest = "(Espesor) StaticResource" />

    <StackLayout Relleno = "5">
        <StackLayout Orientación = "Horizontal">
            <Etiqueta Texto = "{Binding Apellidos, StringFormat = '{0}'}"
                Tamaño de fuente = "Grande" />

            <Etiqueta Texto = "{Binding FirstName} Encuadernación"
                Tamaño de fuente = "Grande" />

            <Etiqueta Texto = "{Binding MiddleName}"
                Tamaño de fuente = "Grande" />
        </StackLayout>
    <BoxView Color = "Acento"
        HeightRequest = "2" />

    <Imagen Fuente = "0 La unión PhotoFilename" />

    <Etiqueta Texto = "{Binding GradePointAverage, StringFormat = 'GPA = {0:F2}'}"
        HorizontalTextAlignment = "Centrar" />

```

```
</ StackLayout >
</ Cuadrícula >
</ ContentView >
```

Ya hemos visto las capturas de pantalla.

Más adjunto propiedades enlazables

propiedades enlazables adjuntos también se pueden configurar en XAML y colocadas con una Estilo. Para ver cómo funciona esto, vamos a examinar una clase llamada `CartesianLayout` que imita una de dos dimensiones, de cuatro cuadrantes sistema de coordenadas cartesianas. Esta disposición permite utilizar `BoxView` para dibujar líneas especificando relativa coordenadas X e Y que van de -1 a 1 con un espesor de línea en particular en unidades de dispositivo.

`CartesianLayout` deriva de Disposición `<BoxView>`, por lo que se limita a los niños de ese tipo. Esta disposición no tiene mucho sentido con otros tipos de elementos. La clase comienza definiendo tres propiedades enlazables adjuntos y estático `Conjunto` y Obtener métodos:

```
espacio de nombres Xamarin.FormsBook.Toolkit
{
    clase pública CartesianLayout : Diseño < BoxView >
    {
        sólo lectura estática pública BindableProperty Point1Property =
            BindableProperty .CreateAttached ( "Punto 1" ,
                tipo de ( Punto ),
                tipo de ( CartesianLayout ),
                nuevo Punto () );

        sólo lectura estática pública BindableProperty Point2Property =
            BindableProperty .CreateAttached ( "Point2" ,
                tipo de ( Punto ),
                tipo de ( CartesianLayout ),
                nuevo Punto () );

        sólo lectura estática pública BindableProperty ThicknessProperty =
            BindableProperty .CreateAttached ( "Espesor" ,
                tipo de ( Doble ),
                tipo de ( CartesianLayout ),
                1.0); // igual al doble de forma explícita!

        public static void valor nominal 1 ( bindableObject enlazable, Punto punto )
        {
            bindable.SetValue (Point1Property, punto);
        }

        public static Punto GetPoint1 ( bindableObject enlazable )
        {
            regreso ( Punto ) Bindable.GetValue (Point1Property);
        }

        public static void setpoint 2 ( bindableObject enlazable, Punto punto )
        {
            bindable.SetValue (Point2Property, punto);
        }
    }
}
```

```
}

public static Punto GetPoint2 ( bindableObject enlazable)
{
    regreso ( Punto ) Bindable.GetValue ( Point2Property );
}

public static void SetThickness ( bindableObject enlazable, doble espesor)
{
    bindable.SetValue ( ThicknessProperty, espesor );
}

public static double GetThickness ( bindableObject enlazable)
{
    regreso ( doble ) Bindable.GetValue ( ThicknessProperty );
}

}

...
```

Al igual que con cualquier propiedades adjuntas se definen en una presentación, se debe invalidar la disposición cada vez que un adjuntas cambios de propiedades que podrían afectar a la disposición. Esta `PropertyChanged` manejador utiliza el Nombre de la propiedad propiedad de la propiedad enlazable para evitar faltas de ortografía:

```
espacio de nombres Xamarin.FormsBook.Toolkit
{
    clase pública CartesianLayout : Diseño < BoxView >
    {
        ...
        // monitor PropertyChanged eventos para los elementos de la colección de los niños.
        protegido override void OnAdded ( BoxView boxView )
        {
            base .OnAdded ( boxView );
            boxView.PropertyChanged += OnChildPropertyChanged;
        }

        protegido override void OnRemoved ( BoxView boxView )
        {
            base .OnRemoved ( boxView );
            boxView.PropertyChanged -= OnChildPropertyChanged;
        }

        vacío OnChildPropertyChanged ( objeto remitente, PropertyChangedEventArgs args )
        {
            Si ( Args.PropertyName == Point1Property.PropertyName ||
                args.PropertyName == Point2Property.PropertyName ||
                args.PropertyName == ThicknessProperty.PropertyName )
            {
                InvalidateLayout ();
            }
        }
        ...
    }
}
```

los OnSizeRequest override requiere que al menos una de las dimensiones se constreñidos y solicita un tamaño que es cuadrado:

```
espacio de nombres Xamarin.FormsBook.Toolkit
{
    clase pública CartesianLayout : Diseño < BoxView >
    {
        ...
        ...
        protected override SizeRequest OnSizeRequest ( doble widthConstraint,
                                                       doble heightConstraint)
        {
            Si (Doble .IsInfinity (widthConstraint) && Doble .IsInfinity (heightConstraint))
                arrojar nueva InvalidOperationException (
                    "CartesianLayout requiere al menos una dimensión a ser limitada." );

            // Que sea cuadrado!
            doble = mínimos Mates .min (widthConstraint, heightConstraint);
            return new SizeRequest ( nuevo tamaño (Mínimo, mínimo));
        }
        ...
    }
}
```

Sin embargo, el diseño resultante *no* ser cuadrado si tiene por defecto HorizontalOptions y Vertical- opciones ajustes de Llenar.

los layoutChildren anulación llama a un método que contiene las matemáticas para traducir el Punto 1, Point2, y Espesor propiedades en una Rectángulo adecuado para una Diseño llamada. los Diseño siempre hace que la llamada BoxView como una línea horizontal situado a medio camino entre Punto 1 y Point2. los Rotación propiedad, entonces gira el BoxView para coincidir con los puntos. La matemática es un poco más compleja que la alternativa (la colocación de la BoxView de modo que comienza en un punto, y luego girar el BoxView para que cumpla con el otro punto), pero este método no requiere ajuste de la anchorX y anchorY propiedades:

```
espacio de nombres Xamarin.FormsBook.Toolkit
{
    clase pública CartesianLayout : Diseño < BoxView >
    {
        ...
        ...
        protected void override layoutChildren ( doble X, doble Y, doble anchura, doble altura)
        {
            para cada ( Ver niño en Niños)
            {
                Si (! Child.isVisible)
                    continuar ;

                doble ángulo;
                Rectángulo fuera = GetChildBounds (niño, x, y, anchura, altura, fuera ángulo);

                // Coloque al niño.
                child.Layout (límites);
            }
        }
    }
}
```

```

        // Girar el niño.
        child.Rotation = ángulo;
    }

}

protegido Rectángulo (GetChildBounds Ver niño,
    doble X, doble Y, doble anchura, doble altura,
    doble a cabo ángulo)
{
    // Obtener información de coordenadas del sistema.
    Punto coordCenter = nuevo Punto (X + ancho / 2, y + altura / 2);
    doble unitlength = Mates .min (anchura, altura) / 2;

    // Información niño.
    Punto point1 = GetPoint1 (niño);
    Punto punto2 = GetPoint2 (niño);
    doble espesor = GetThickness (niño);
    doble longitud = unitlength * Mates .Sqr ( Mates .Pow (point2.X - point1.X, 2) +
                                                Mates .Pow (point2.Y - point1.Y, 2));

    // Calcular límites niño.
    Punto centerChild = nuevo Punto ((Point1.X + point2.X) / 2,
                                      (Point1.Y + point2.Y) / 2);

    doble Xchild = coordCenter.X + unitlength * centerChild.X - longitud / 2;
    doble yChild = coordCenter.Y - unitlength * centerChild.Y - espesor / 2;
    Rectángulo = límites nuevo Rectángulo (Xchild, yChild, longitud, espesor);
    ángulo = 180 / Mates .PI * Mates .Atan2 (point1.Y - point2.Y,
                                                point2.X - point1.X);

    regreso límites;
}

```

Puede establecer las propiedades enlazables adjuntas en XAML e incluso en una Estilo, sino porque se requiere que el nombre de la clase al hacer referencia a propiedades enlazables adjuntas, las propiedades también requieren la declaración de espacio de nombres XML.. los **UnitCube** programa dibuja el contorno de un cubo 3D:

```
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
      xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
      xmlns: kit de herramientas =
          "clr-espacio de nombres: Xamarin.FormsBook.Toolkit; montaje = Xamarin.FormsBook.Toolkit"
      x: Class = "UnitCube.UnitCubePage ">

    < kit de herramientas: CartesianLayout Color de fondo = "Amarillo"
        HorizontalOptions = "Centrar"
        VerticalOptions = "Centrar" >

    < Herramienta: CartesianLayout.Resources >
        < ResourceDictionary >
            < Estilo x: Key = "baseStyle" Tipo de objetivo = "BoxView" >
                < Setter Propiedad = "Color" Valor = "Azul" />
                < Setter Propiedad = "kit de herramientas: CartesianLayout.Thickness" Valor = "3" />
            </ Estilo >
        </ ResourceDictionary >
    </ Herramienta: CartesianLayout.Resources >
</ Pagina de contenido >
```

```
< Estilo x: Key = " hiddenStyle " Tipo de objetivo = " BoxView "
    Residencia en = " {} StaticResource baseStyle " >
    < Setter Propiedad = " Opacidad " Valor = " 0.25 " />
</ Estilo >

<! - estilo implícita. ->
< Estilo Tipo de objetivo = " BoxView "
    Residencia en = " {} StaticResource baseStyle " />

</ ResourceDictionary >
</ Herramienta: CartesianLayout.Resources >

<! - Tres bordes "ocultos" primeros en el fondo ->
<! - bordes traseros ->
< BoxView kit de herramientas: CartesianLayout.Point1 = " 0.25, 0.75 "
    kit de herramientas: CartesianLayout.Point2 = " 0.25, -0.25 "
    Estilo = " {} StaticResource hiddenStyle " />

< BoxView kit de herramientas: CartesianLayout.Point1 = " 0.25, -0.25 "
    kit de herramientas: CartesianLayout.Point2 = " -0.75, -0.25 "
    Estilo = " {} StaticResource hiddenStyle " />

<! - Frente al borde posterior ->
< BoxView kit de herramientas: CartesianLayout.Point1 = " 0.5, -0.5 "
    kit de herramientas: CartesianLayout.Point2 = " 0.25, -0.25 "
    Estilo = " {} StaticResource hiddenStyle " />

<! - bordes delanteros ->
< BoxView kit de herramientas: CartesianLayout.Point1 = " -0.5, 0.5 "
    kit de herramientas: CartesianLayout.Point2 = " 0.5, 0.5 " />

< BoxView kit de herramientas: CartesianLayout.Point1 = " 0.5, 0.5 "
    kit de herramientas: CartesianLayout.Point2 = " 0.5, -0.5 " />

< BoxView kit de herramientas: CartesianLayout.Point1 = " 0.5, -0.5 "
    kit de herramientas: CartesianLayout.Point2 = " -0.5, -0.5 " />

< BoxView kit de herramientas: CartesianLayout.Point1 = " -0.5, -0.5 "
    kit de herramientas: CartesianLayout.Point2 = " -0.5, 0.5 " />

<! - bordes traseros ->
< BoxView kit de herramientas: CartesianLayout.Point1 = " -0.75, 0.75 "
    kit de herramientas: CartesianLayout.Point2 = " 0.25, 0.75 " />

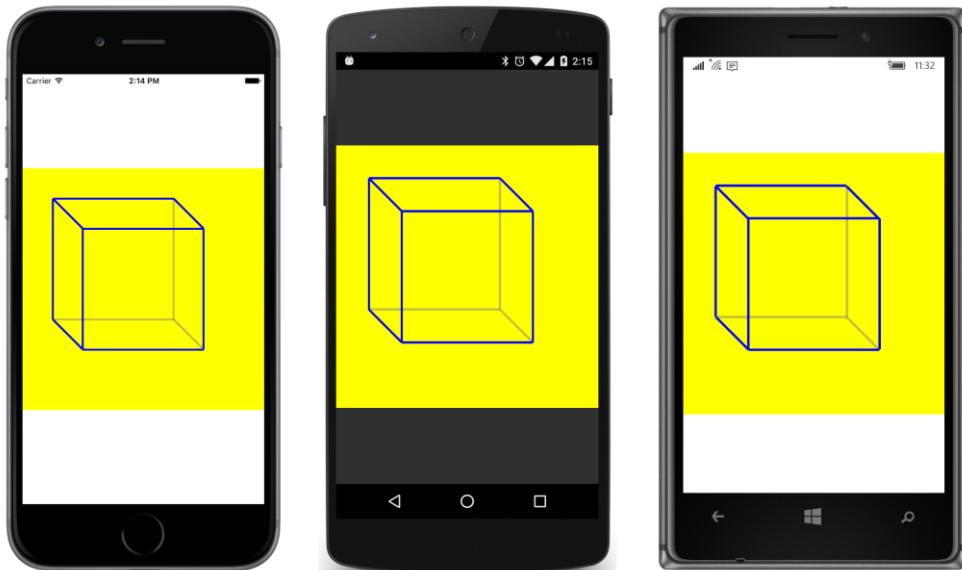
< BoxView kit de herramientas: CartesianLayout.Point1 = " -0.75, -0.25 "
    kit de herramientas: CartesianLayout.Point2 = " -0.75, 0.75 " />

<! - Frente a los bordes traseros ->
< BoxView kit de herramientas: CartesianLayout.Point1 = " -0.5, 0.5 "
    kit de herramientas: CartesianLayout.Point2 = " -0.75, 0.75 " />

< BoxView kit de herramientas: CartesianLayout.Point1 = " 0.5, 0.5 "
    kit de herramientas: CartesianLayout.Point2 = " 0.25, 0.75 " />
```

```
< BoxView kit de herramientas: CartesianLayout.Point1 = "-0.5, -0.5 "
  kit de herramientas: CartesianLayout.Point2 = "-0.75, -0.25 " >
</! kit de herramientas: CartesianLayout >
</! Pagina de contenido >
```

El fondo “líneas” se dibujan con una Opacidad valor que los hace parecer como si son vistos a través de un lado translúcida:



Diseño y LayoutTo

VisualElement define una colección de propiedades de transformación. Estos son AnchorX, anchorY, rotación, rotationX, RotationY, Escala, TranslationX, y TranslationY, y que no afecten a la disposición en absoluto. En otras palabras, establecer estas propiedades no genera llamadas a InvalidateMeasure o InvalidateLayout. tamaños de elementos devueltos desde GetSizeRequest no se ven afectados por estas propiedades. Los Diseño llamar tamaños y elementos posiciones como si estas propiedades no existen.

Esto significa que se pueden animar estas propiedades sin generar un montón de ciclos de diseño. Los traducir a, scaleTo, RotateTo, RotateXTo, y RotateYTo métodos de animación definidos como métodos de extensión en ViewExtensions son totalmente independientes de diseño.

Sin embargo, ViewExtensions también define un método llamado LayoutTo que hace que las llamadas a la animación Diseño método. Esto se traduce en el cambio del tamaño del diseño o la posición del elemento con respecto a su matriz y el establecimiento de nuevos valores del elemento de Límites, X, Y, Ancho, y Altura propiedades.

Utilizando LayoutTo por lo tanto requiere el ejercicio de algunas precauciones.

Por ejemplo, supongamos que un elemento es un niño de una StackLayout. Cuando StackLayout para crear una LayoutTo método llamar, se llamará Diseño en ese elemento con el tamaño y la posición en una localización particular con respecto a sí mismo. Suponga que su programa llama a continuación, LayoutTo en ese elemento para darle un nuevo tamaño y posición. Los StackLayout no sé nada de eso, así que si el StackLayout se somete a otro ciclo de diseño, se moverá el elemento de nuevo a donde cree que debería ser. Si aún necesita el elemento de estar en un lugar distinto de aquél en el StackLayout piensa que debería ser, es posible que desee conectar un controlador a la LayoutChanged caso de la StackLayout y llama Diseño o ejecutar el LayoutTo animación en ese elemento nuevo.

Otro problema está en marcha una LayoutTo animación en un diseño con muchos niños. Está permitido, por supuesto, pero hay que tener en cuenta que la disposición obtendrá numerosas llamadas a su Diseño método, y por lo tanto también su layoutChildren método, mientras que la animación está en curso. Para cada una de estas llamadas a su Diseño-Niños anulación, la clase de diseño va a tratar de exponer todos sus hijos (y, por supuesto, algunos de estos niños podría haber otros diseños con los niños), y la animación podría llegar a ser bastante agitado.

Pero se puede utilizar la relación entre el LayoutTo animación y el Diseño método para poner en práctica algunos efectos interesantes. Un elemento debe tener su Diseño método llamado a ser visible en la pantalla, pero llamar LayoutTo satisface ese requisito.

Aquí hay una clase que deriva de CartesianLayout, llamado AnimatedCartesianLayout. Se define dos propiedades enlazables (no unido propiedades enlazables) para gobernar la animación, y en lugar de llamar Diseño y el establecimiento de la Rotación propiedad, se llama LayoutTo y (opcionalmente) RotateTo:

```
espacio de nombres Xamarin.FormsBook.Toolkit
{
    clase pública AnimatedCartesianLayout : CartesianLayout
    {
        sólo lectura estática pública BindableProperty AnimationDurationProperty =
            BindableProperty .Crear(
                "AnimatedDuration",
                tipo de ( En t ),
                tipo de ( AnimatedCartesianLayout ),
                1000);

        public int Duración de la animación
        {
            conjunto {EstablecerValor (AnimationDurationProperty, valor);}
            obtener {regreso ( En t ) GetValue (AnimationDurationProperty);}
        }

        sólo lectura estática pública BindableProperty AnimateRotationProperty =
            BindableProperty .Crear(
                "AnimateRotation",
                tipo de ( bool ),
                tipo de ( AnimatedCartesianLayout ),
                cierto );

        public bool AnimateRotation
        {
            conjunto {EstablecerValor (AnimateRotationProperty, valor);}
        }
    }
}
```

```

        obtener { regreso ( bool ) GetValue (AnimateRotationProperty); }
    }

    protegido override void layoutChildren ( doble X, doble Y, doble anchura, doble altura)
    {
        para cada ( Ver niño en Niños )
        {

            Si ( ! Child.isVisible )
                continuar ;

            doble ángulo;
            Rectángulo fuera = GetChildBounds ( niño, x, y, anchura, altura, fuera ángulo );

            // Coloque al niño.
            Si ( child.Bounds.Equals ( nuevo Rectángulo ( 0, 0, -1, -1 ) ) )
            {
                child.Layout ( nuevo Rectángulo ( X + ancho / 2, y + altura / 2, 0, 0 ) );
            }
            child.LayoutTo ( límites, ( uint )Duración de la animación );

            // Girar el niño.
            Si ( AnimateRotation )
            {
                child.RotateTo ( ángulo, ( uint )Duración de la animación );
            }
            más
            {
                child.Rotation = ángulo;
            }
        }
    }
}

```

La única parte difícil involucra a un niño que aún no ha recibido su primera Diseño llamada. Los Límites bienes de un niño tal es el rectángulo (0, 0, -1, -1), y el LayoutTo animación utilizará ese valor como el punto de partida para la animación. En ese caso, el layoutChildren Primer método de llamadas Diseño para posicionar el niño en el centro y para darle un tamaño de (0, 0).

los **AnimatedUnitCube** programa tiene un archivo XAML casi idéntica a la **UnitCube** programa, pero con una **AnimatedCartesianLayout** con una duración de la animación de 3 segundos:

```

< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    xmlns: kit de herramientas =
        "clr-espacio de nombres: Xamarin.FormsBook.Toolkit; montaje = Xamarin.FormsBook.Toolkit "
    x: Class = " AnimatedUnitCube.AnimatedUnitCubePage " >

    < kit de herramientas: AnimatedCartesianLayout Color de fondo = " Amarillo "
        Duración de la animación = " 3000 "
        HorizontalOptions = " Centrar "
        VerticalOptions = " Centrar " >

    < Herramienta: AnimatedCartesianLayout.Resources >

```

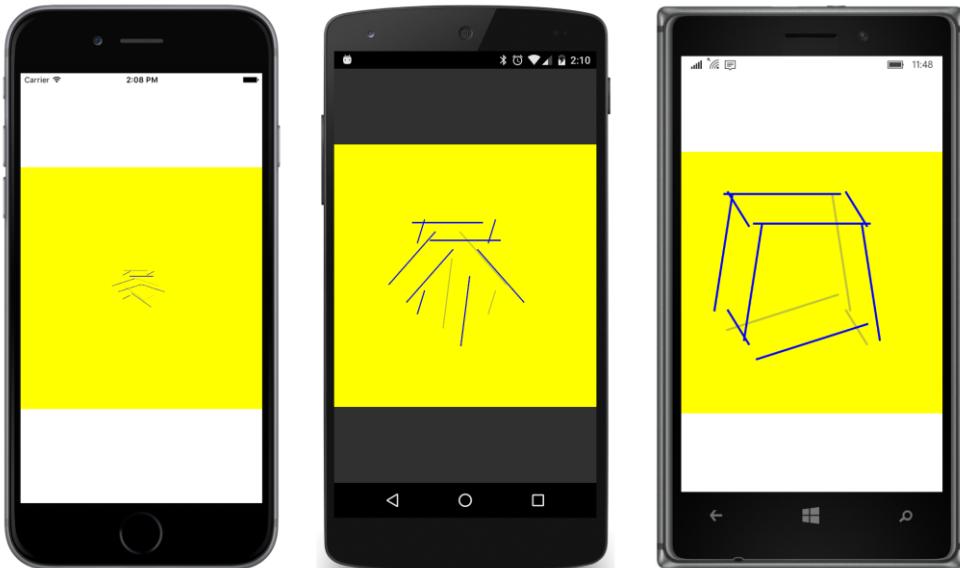
```
< ResourceDictionary >
    < Estilo x: Key = " baseStyle " Tipo de objetivo = " BoxView " >
        < Setter Propiedad = " Color " Valor = " Azul " />
        < Setter Propiedad = " kit de herramientas: CartesianLayout.Thickness " Valor = " 3 " />
    </ Estilo >

    < Estilo x: Key = " hiddenStyle " Tipo de objetivo = " BoxView " >
        Residencia en = " {} StaticResource baseStyle "
        < Setter Propiedad = " Opacidad " Valor = " 0.25 " />
    </ Estilo >

    <! - estilo implica. ->
    < Estilo Tipo de objetivo = " BoxView " >
        Residencia en = " {} StaticResource baseStyle "
    </ Estilo >

</ ResourceDictionary >
</ Herramienta: AnimatedCartesianLayout.Resources >
...
</ kit de herramientas: AnimatedCartesianLayout >
</ Pagina de contenido >
```

Las siguientes capturas de pantalla muestran la progresión de izquierda a derecha casi hasta el punto en que el cubo es completa:



Dependiendo de cómo se definen, algunas de las líneas horizontales no se giran en absoluto, mientras que otros (los que están en la parte inferior, por ejemplo) deben girarse 180 grados.

Como se sabe, las interfaces de usuario se han vuelto más animada y dinámica en los últimos años, por lo que la exploración de diversas técnicas que son posibles mediante el uso de `LayoutTo` más bien que Diseño puede convertirse en un área completamente nueva para los programadores de aventura a seguir.

capítulo 27

procesadores personalizados

En el núcleo de Xamarin.Forms es algo que podría parecer magia: la capacidad de un solo elemento, como Botón para que aparezca como un botón nativo bajo los sistemas operativos iOS, Android y Windows. En este capítulo, verá cómo en las tres plataformas cada elemento en Xamarin.Forms es apoyado por una clase especial conocido como *renderizador*. Por ejemplo, el Botón clase en Xamarin.Forms es apoyada por varias clases en las diversas plataformas, cada uno nombrado ButtonRenderer.

La buena noticia es que también se puede escribir sus propios procesadores, y este capítulo le mostrará cómo. Sin embargo, tenga en cuenta que la escritura de procesadores personalizados es un gran tema, y este capítulo sólo puede empezar.

Escribiendo procesadores personalizados no es tan fácil como escribir una aplicación Xamarin.Forms. Tendrá que estar familiarizado con los iOS individuales, Android y plataformas de Windows en tiempo de ejecución. Pero, obviamente, es una técnica poderosa. De hecho, algunos desarrolladores piensan en el valor final de Xamarin.Forms como proporcionar un marco estructurado en el que escribir procesadores personalizados.

La jerarquía de clases completa

En el capítulo 11, “La infraestructura enlazable”, que vio un programa llamado **ClassHierarchy** que muestra la jerarquía de clases Xamarin.Forms. Sin embargo, ese programa sólo muestra los tipos en el **Xamarin.Forms.Core** y **Xamarin.Forms.Xaml** conjuntos, que son los tipos que utiliza generalmente una aplicación Xamarin.Forms.

Xamarin.Forms también contiene conjuntos adicionales asociados con cada plataforma. Estas asambleas desempeñan un papel crucial al proporcionar el apoyo plataforma para Xamarin.Forms, incluyendo todos los procesadores.

Es probable que ya familiarizados con los nombres de estos conjuntos de verlas en el **Referencia secciones de los** diversos proyectos en su solución Xamarin.Forms:

- **Xamarin.Forms.Platform** (muy pequeña)
- **Xamarin.Forms.Platform.iOS**
- **Xamarin.Forms.Platform.Android**
- **Xamarin.Forms.Platform.UAP**
- **Xamarin.Forms.Platform.WinRT** (mayor de los dos siguientes en esta lista)
- **Xamarin.Forms.Platform.WinRT.Tablet**

- **Xamarin.Forms.Platform.WinRT.Phone**

En esta discusión, éstas se denominan colectivamente como la *conjuntos de plataforma*.

¿Es posible escribir una aplicación Xamarin.Forms que muestra una jerarquía de clases de los tipos de estos conjuntos de plataforma?

¡Sí! Sin embargo, si se restringe a sí mismo a examinar únicamente los montajes cargados normalmente con una aplicación, y esto es sin duda el más simple enfoque a continuación, una aplicación sólo puede mostrar los tipos de los conjuntos que forman parte de esa aplicación. Por ejemplo, sólo puede visualizar los tipos en el

Xamarin.Forms.Platform.iOS montaje con un programa Xamarin.Forms ejecuta en iOS, y lo mismo para los otros conjuntos.

Pero todavía hay un problema: Como se puede recordar, el original **ClassHierarchy** programa se inició mediante la obtención de .NET Montaje objetos para el **Xamarin.Forms.Core** y **Xamarin.Forms.Xaml** ensamblajes basados en dos clases (Ver y extensiones) que sabía que en esos dos conjuntos:

```
tipo de ( Ver ).GetTypeInfo ().Asamblea
tipo de ( extensiones ).GetTypeInfo ().Asamblea
```

Sin embargo, la biblioteca de clases de una aplicación portátil Xamarin.Forms no tiene acceso directo a los conjuntos de plataforma. Los conjuntos de plataforma se hace referencia sólo por los proyectos de aplicación. Esto significa que una biblioteca de clases portátil Xamarin.Forms no puede utilizar un código similar para obtener una referencia a la unidad de la plataforma. Esto no funcionará:

```
tipo de ( ButtonRenderer ).GetTypeInfo ().Asamblea
```

Sin embargo, estos conjuntos de plataforma se cargan cuando la aplicación se ejecuta, por lo que el PCL en lugar puede obtener Montaje Objetos para los conjuntos de plataforma basado en el nombre de ensamblado. Los **PlatformClassHierarchy** programa comienza de esta manera:

```
pública clase parcial PlatformClassHierarchyPage : Pagina de contenido
{
    pública PlatformClassHierarchyPage ()
    {
        InitializeComponent ();
        Lista < TypeInformation > = ClassList nuevo Lista < TypeInformation > 0;

        cuerda [] AssemblyNames = Dispositivo .OnPlatform (
            iOS: nueva cadena [] { "Xamarin.Forms.Platform.iOS" },
            Androide: nueva cadena [] { "Xamarin.Forms.Platform.Android" },
            WinPhone: nueva cadena [] { "Xamarin.Forms.Platform.UAP" ,
                "Xamarin.Forms.Platform.WinRT",
                "Xamarin.Forms.Platform.WinRT.Tablet",
                "Xamarin.Forms.Platform.WinRT.Phone" }
        );
        para cada ( cuerda assemblyName en assemblyNames )
        {
            tratar
```

```

    {
        Montaje montaje = Montaje .Carga( nuevo AssemblyName (AssemblyName));
        GetPublicTypes (montaje, classList);
    }
    captura
    {
    }
}
...
}

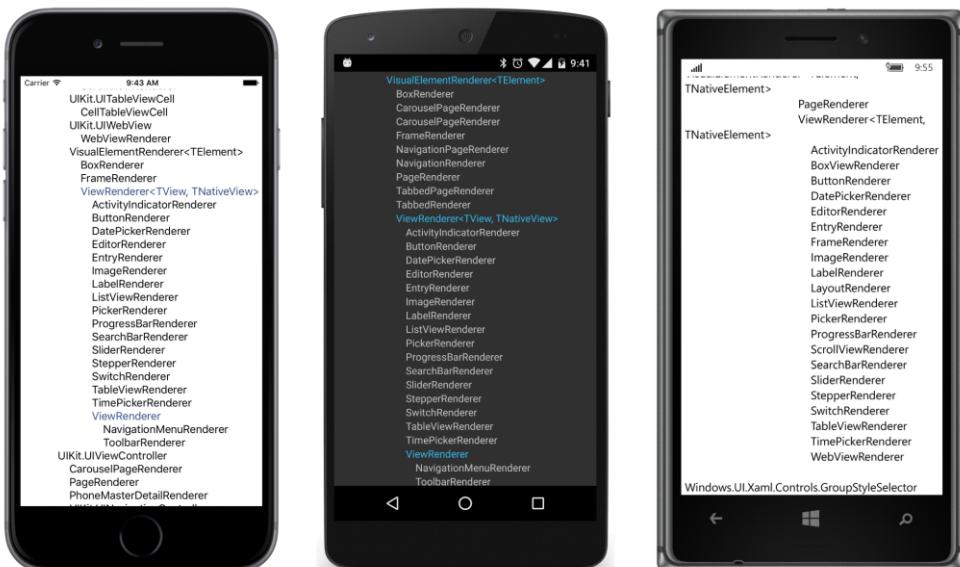
```

Y desde allí el **PlatformClassHierarchy** programa es el mismo que el original **ClassHierarchy** programa.

Como se puede ver, el para cada bucle obtiene la Montaje objeto de la estática **Assembly.Load**

método. Sin embargo, no hay una forma sencilla para el programa para determinar si se está ejecutando bajo la plataforma universal de Windows o una de las otras plataformas de Windows en tiempo de ejecución, por lo que si **Device**-**OnPlatform** indica que se trata de la **WinPhone** dispositivo, el programa trata de los cuatro montajes y usos tratar y captura simplemente ignorar las que no funcionan.

Algunos de los nombres-y clase particularmente los nombres de clase completos para las clases fuera del montaje son un poco demasiado largo para la visualización vertical y envolver con torpeza, pero aquí es parte de la pantalla en las tres plataformas. Cada uno ha sido desplazado a la parte de la jerarquía de clases que comienza con el genérico **ViewRenderer** clase. Esta es generalmente la clase se le derivan de crear sus propios procesadores personalizados:



Observe los parámetros genéricos para el **ViewRenderer** clase llamada ya sea **TView** y **TNativeView**, o **TElement** y **TNativeElement**: Como se verá, **TView** o **TElement** es el elemento **Xamarin.Forms**,

como Botón, mientras TNativeView o TNativeElement es el control nativo para ese Botón.

Aunque el **PlatformClassHierarchy** programa no indica esto, las restricciones para el ViewRenderer parámetros genéricos son dependientes de la plataforma:

- En iOS:
 - TView está limitada a Xamarin.Forms.View
 - TNativeView está limitada a UIKit.UIView
- En Android:
 - TView está limitada a Xamarin.Forms.View
 - TNativeView está limitada a Android.Views.View
- En las plataformas Windows:
 - TElement está limitada a Xamarin.Forms.View
 - TNativeElement está limitada a Windows.UI.Xaml.FrameworkElement

Para escribir un procesador personalizado, se derive una clase de ViewRenderer. Para dar cabida a todas las plataformas, debe implementar el procesador de iOS mediante el uso de una clase que se deriva de UIView, implementar el procesador de Android con una clase que deriva de View, e implementar un procesador para las plataformas de Windows con una clase que deriva de FrameworkElement.

¡Vamos a intentarlo!

Hola, a medida renderizadores!

los HelloRenderers programa sobre todo demuestra la sobrecarga necesaria para escribir extracción de grasas simples. El programa define una nueva Ver derivado llamado HelloView Con ello se pretende mostrar una cadena simple fija de texto. Aquí está el archivo de HelloView.cs completas en el HelloRenderers Portátil proyecto de biblioteca de clases:

utilizando Xamarin.Forms;

```
espacio de nombres HelloRenderers
{
    clase pública HelloView : Ver
    {
    }
}
```

¡Eso es! Sin embargo, tenga en cuenta que la clase se define como público. A pesar de que se podría pensar que esta clase sólo se hace referencia en el PCL, ese no es el caso. Debe ser visible para los conjuntos de plataforma.

los **HelloRenderers** PCL es tan simple que ni siquiera se molesta con una clase de página. En su lugar, se crea una instancia y muestra una HelloView objeto centrado en la página correcta en el archivo App.cs:

```
espacio de nombres HelloRenderers
{
    clase pública Aplicación : Solicitud
    {
        público App ()
        {
            MainPage = nuevo Pagina de contenido
            {
                content = nuevo HelloView
                {
                    VerticalOptions = LayoutOptions .Centrar,
                    HorizontalOptions = LayoutOptions .Centrar
                };
            }
            ...
        };
    }
}
```

Sin ningún otro código, este programa funciona muy bien, pero en realidad no ver el HelloView objeto en la pantalla, porque es sólo una visión transparente en blanco. Lo que necesitamos son algunos procesadores de la plataforma para HelloView.

Cuando una aplicación Xamarin.Forms se pone en marcha, Xamarin.Forms utiliza .NET reflexión para buscar a través de los diversos conjuntos que componen la aplicación, en busca de atributos de ensamblado con nombre Exportar-Renderizador. Un ExportRenderer atributo indica la presencia de un intérprete personalizado que puede proporcionar soporte para un elemento Xamarin.Forms.

los **HelloRenderers.iOS** proyecto contiene el archivo siguiente HelloViewRenderer.cs, que se muestra en su totalidad. Observe la ExportRenderer atribuir justo debajo de la utilizando directivas. Debido a que este es un atributo de montaje, debe ser fuera de una espacio de nombres declaración. Este particular ExportRenderer atributo básicamente dice: "El HelloView clase está soportado por un procesador de tipo HelloViewRenderer":

```
utilizando Xamarin.Forms;
utilizando Xamarin.Forms.Platform.iOS;

utilizando UIKit;

utilizando HelloRenderers;
utilizando HelloRenderers.iOS;

[ montaje : ExportRenderer ( tipo de ( HelloView ), tipo de ( HelloViewRenderer ))]

espacio de nombres HelloRenderers.iOS
{
    clase pública HelloViewRenderer : ViewRenderer < HelloView , UILabel >
    {
        protected void override OnElementChanged ( ElementChangedEventArgs < HelloView > args )
        {
```

```

base.OnElementChanged(args);

Si (== control nulo )
{
    UILabel etiqueta = nuevo UILabel
    {
        text = "Hola desde iOS!" ,
        = tipo de letra UIFont.SystemFontOfSize(24)
    };
}

SetNativeControl(etiqueta);
}
}
}

```

La definición de la HelloViewRenderer clase sigue la ExportRenderer atributo. La clase debe ser pública. Se deriva de la genérica ViewRenderer clase. Los dos parámetros genéricos se nombran TView, que es la clase Xamarin.Forms, y TNativeView, que es la clase en este caso particular que es nativa de iOS.

En iOS, una clase que muestra el texto es `UILabel` en el UIKit espacio de nombres, y eso es lo que se utiliza aquí. Los dos argumentos genéricas a `ViewRenderer` básicamente decir "Un `HelloView` objeto es en realidad representa como un iOS `UILabel` objeto."

La un trabajo esencial para una ViewRenderer derivado es para anular el OnElementChanged método. Este método se llama cuando una HelloView Se crea objeto, y su trabajo es crear un control nativo para la prestación del HelloView objeto.

Los `OnElementChanged` override comienza comprobando el `Controlar` propiedad de que la clase hereda de `ViewRenderer`. Esta `Controlar` propiedad se define por `ViewRenderer` a ser de tipo `TNa-
tiveView`, por lo que en `HelloViewRenderer` que es de tipo `UILabel`. La primera vez que `OnElementChanged` se llama, este `Controlar` propiedad
será nulo. Los `UILabel` objeto debe ser creado. Esto es lo que hace el método, asignándole un texto y un tamaño de fuente. Ese `UILabel` método
se pasa a continuación a la

`SetNativeControl` método. A continuación, la `Controlar` propiedad estará presente `UILabel` objeto.

los utilizando directivas en la parte superior del archivo se dividen en tres grupos:

- los utilizando Directiva para la Xamarin.Forms Se requiere espacio de nombres para el ExportRenderer atribuir, mientras Xamarin.Forms.Platform.iOS se requiere para el ViewRenderer clase.
 - el iOS UIKit Se requiere espacio de nombres para UILabel.
 - los utilizando directivas para HelloRenderers y HelloRenderers.iOS Sólo son necesarios para la HelloView y HelloViewRenderer referencias en el ExportRenderer atribuir porque el atributo debe estar fuera del HelloRenderer.iOS bloque de espacio de nombres.

Los dos últimos utilizando directivas son particularmente molesto porque sólo están obligados por un solo propósito. Si desea, puede deshacerse de esos dos utilizando directivas por calificar totalmente los nombres de las clases

dentro de ExportRenderer atributo.

Esto se realiza de la siguiente renderizador. Aquí está el archivo de HelloViewRenderer.cs completas en el **HelloRenderers.Droid** proyecto.

El widget Android de texto que muestra es Vista de texto en el **Android.Widget**

espacio de nombres:

utilizando Xamarin.Forms;

utilizando Xamarin.Forms.Platform.Android;

utilizando Android.Util;

utilizando Android.Widget;

```
[ montaje : ExportRenderer ( tipo de (HelloRenderers. HelloView ),
    tipo de (HelloRenderers.Droid. HelloViewRenderer ))]
```

espacio de nombres HelloRenderers.Droid

```
{
    clase pública HelloViewRenderer : ViewRenderer < HelloView , Vista de texto >
    {
        protected void override OnElementChanged ( ElementChangedEventArgs < HelloView > args)
        {
            base .OnElementChanged (args);

            Si (== control nulo )
            {
                SetNativeControl ( nuevo Vista de texto (Contexto)
                {
                    text = "Hola desde Android!";
                });

                Control.SetTextSize ( ComplexUnitType .Sp, 24);
            }
        }
    }
}
```

Esta **HelloViewRenderer** clase se deriva de la versión para Android de **ViewRenderer**. Los argumentos genéricos para **ViewRenderer** indicar que el **HelloView** clase se apoya en el **Android Vista de texto**

Widget.

Una vez más, en la primera llamada a **OnElementChanged**, el **Controlar** propiedad será nulo. El método debe crear un nativo de **Android Vista de texto** widget y llamada **SetNativeControl**. Para ahorrar un poco de espacio, el recién instanciada Vista de texto objeto se pasa directamente a la **SetNativeControl** método. Observe que el Vista de texto constructor requiere el **Android Contexto** objeto. Esto está disponible como una propiedad de **OnElementChanged**.

Después de la llamada a **SetNativeControl**, el **Controlar** Propiedad Definido por **ViewRenderer** es el widget nativo de **Android**, en este caso el **Vista de texto** objeto. El método utiliza este **Controlar** para llamar a la propiedad

SetTextSize sobre el **Vista de texto** objeto. En **Android**, tamaños de texto se pueden escalar en una variedad de maneras. los

ComplexUnitType.Sp miembro de la enumeración indica “píxeles escalados”, que es compatible con la forma

Xamarin.Forms maneja tamaños de letra para Etiqueta en Android.

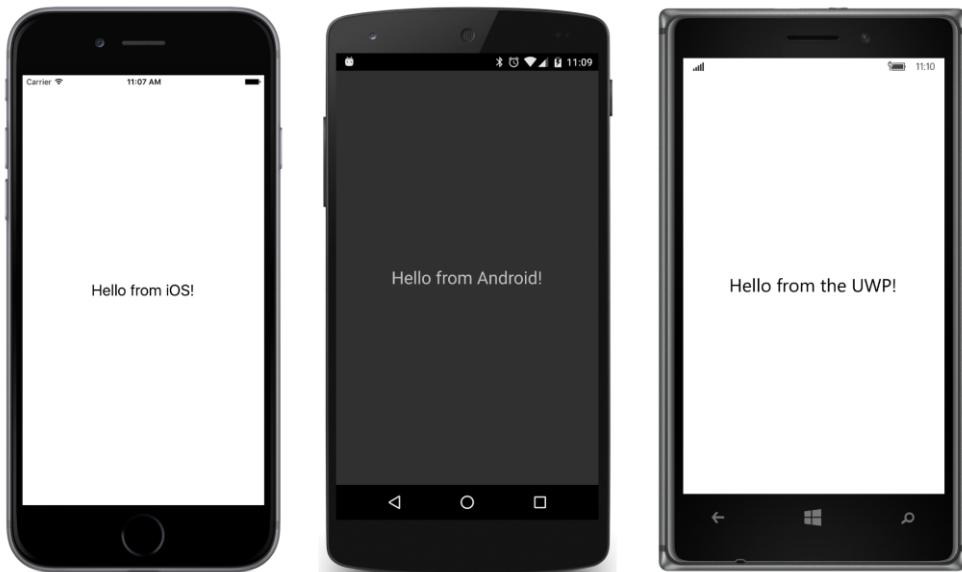
Aquí está la versión de uwp HelloViewRenderer en el **HelloRenderers.UWP** proyecto:

```
utilizando Xamarin.Forms.Platform.UWP;  
  
utilizando Windows.UI.Xaml.Controls;  
  
[ montaje : ExportRenderer ( tipo de (HelloRenderers. HelloView ),  
    tipo de (HelloRenderers.UWP. HelloViewRenderer ))]  
  
espacio de nombres HelloRenderers.UWP  
{  
    clase pública HelloViewRenderer : ViewRenderer < HelloView , Bloque de texto >  
    {  
        protegido override void OnElementChanged ( ElementChangedEventArgs < HelloView > args )  
        {  
            base .OnElementChanged ( args );  
  
            Si ( == control nulo )  
            {  
                SetNativeControl ( nuevo Bloque de texto  
                {  
                    text = "Hola desde el UWP!" ,  
                    FontSize = 24 ,  
                } );  
            }  
        }  
    }  
}
```

En todas las plataformas Windows, el HelloView objeto se representa por un tiempo de ejecución de Windows Bloque de texto en el Windows.UI.Xaml.Controls espacio de nombres.

los HelloViewRenderer clases en el **HelloRenderers.Windows** y **HelloRenderers.WinPhone** proyectos son en su mayoría los mismos a excepción de los espacios de nombres y el texto utilizados para establecer el Texto propiedad de Bloque de texto.

Aquí está el programa que se ejecuta en las tres plataformas estándar:



Observe cómo el texto se centra correctamente a través del uso de la normalidad `HorizontalOptions` y `VerticalOptions` propiedades establecen en el `HelloView` objeto. Sin embargo, no se puede establecer la `HorizontalTextAlignment` y `VerticalTextAlignment` propiedades en `HelloView`. Estas propiedades están definidas por `Etiqueta` y no por `HelloView`.

para activar `HelloView` en una visión de pleno derecho para la visualización de texto, lo que se necesita para empezar a añadir propiedades a la `HelloView` clase.

Vamos a examinar cómo se añaden a las propiedades de extracción de grasas con un ejemplo diferente.

Renderizadores y propiedades

Xamarin.Forms incluye una `BoxView` elemento para la visualización de bloques rectangulares de color. ¿Alguna vez ha deseado tener algo similar para dibujar un círculo, o para que sea más generalizada, una elipse?

Ese es el propósito de `EllipseView`. Sin embargo, como es posible que desee utilizar `EllipseView` en múltiples aplicaciones, que se implementa en el `Xamarin.FormsBook.Platform` bibliotecas, introducidos en el capítulo 20, "asíncrono y el archivo de E / S".

`BoxView` define una propiedad por su propia cuenta, una `Color` propiedad de tipo `Color` -y `EllipseView` puede hacer lo mismo. No necesita propiedades para ajustar la anchura y la altura de la elipse porque hereda `WidthRequest` y `HeightRequest` de `VisualElement`.

Así que aquí está `EllipseView` como se define en la `Xamarin.FormsBook.Platform` proyecto de biblioteca:

```
espacio de nombres Xamarin.FormsBook.Platform
{
}
```

```

clase pública EllipseView : Ver
{
    sólo lectura estática pública BindableProperty ColorProperty =
        BindableProperty .Crear(
            "Color",
            tipo de ( Color ),
            tipo de ( EllipseView ),
            Color .Defecto);

    público Color Color
    {
        conjunto (EstablecerValor (ColorProperty, valor);)
        obtener { regreso ( Color ) GetValue (ColorProperty); }
    }

    protegido anular SizeRequest OnSizeRequest ( doble widthConstraint,
                                                doble heightConstraint)
    {
        return new SizeRequest ( nuevo tamaño (40, 40));
    }
}
}

```

los Color la propiedad consiste simplemente en una definición básica de una propiedad que puede vincularse con ningún controlador PropertyChanged.

La propiedad está definida, pero no parece estar haciendo nada. De alguna manera, la

Color propiedad definida en EllipseView tiene que estar vinculado con una propiedad en el objeto de que el procesador está prestando.

El único otro código en EllipseView es una anulación de OnSizeRequest para fijar un tamaño predeterminado de la elipse, lo mismo que BoxView.

Vamos a comenzar con la plataforma Windows. Resulta que un procesador de Windows para EllipseView es más simple que los prestadores de iOS y Android.

Como se recordará, el **Xamarin.FormsBook.Platform** solución creada en el capítulo 20 tiene una facilidad para permitir el intercambio de código entre las diferentes plataformas de Windows: La **Xamarin.FormsBook.Platform.UWP**

biblioteca, el **Xamarin.FormsBook.Platform.Windows** biblioteca y la **Xamarin.FormsBook.Platform.WinPhone** biblioteca todos tienen referencias al **Xamarin.FormsBook.Platform.WinRT** biblioteca, que no es una biblioteca en absoluto, sino en realidad un proyecto compartido. Este proyecto compartido es donde el Elipse-

ViewRenderer clase para todas las plataformas de Windows puede residir.

En las plataformas Windows, una EllipseView puede ser dictada por un elemento nativo de Windows llamado

Elipse en el **Windows.UI.Xaml.Shapes** espacio de nombres, porque Elipse satisface los criterios de derivados de **Windows.UI.Xaml.FrameworkElement**.

los Elipse se especifica como el segundo argumento genérico para la ViewRenderer clase. Dado que este archivo es compartido por todas las plataformas Windows, necesita algunas directivas de preprocessamiento para incluir el espacio de nombre correcto para el ExportRendererAttribute y ViewRenderer clases:

utilizando **System.ComponentModel**;

```

utilizando Windows.UI.Xaml.Media;
utilizando Windows.UI.Xaml.Shapes;

# Si WINDOWS_UWP
utilizando Xamarin.Forms.Platform.UWP;
#ELSE
utilizando Xamarin.Forms.Platform.WinRT;
# terminara si

[ montaje : ExportRenderer ( tipo de (Xamarin.FormsBook.Platform. EllipseView ),
                           tipo de (Xamarin.FormsBook.Platform.WinRT. EllipseViewRenderer ))]

espacio de nombres Xamarin.FormsBook.Platform.WinRT
{
    clase pública EllipseViewRenderer : ViewRenderer < EllipseView , Ellipse >
    {
        protected void override OnElementChanged ( ElementChangedEventArgs < EllipseView > args)
        {
            base .OnElementChanged (args);

            Si ( == control nulo )
            {
                SetNativeControl ( nuevo Ellipse ());
            }

            Si (Args.NewElement!= nulo )
            {
                SetColor ();
            }
        }
        ...
    }
}

```

Como era de esperar por el momento, la `OnElementChanged` anular primero comprueba si el `Control` propiedad es nulo, y si es así, se crea el objeto nativo, en este caso una Elipse, y lo pasa a `SetNativeControl`. A continuación, la `Control` propiedad se establece en este Elipse objeto.

Esta `OnElementChanged` override también contiene algo de código adicional que implica la `ElementChangedEventArgs` argumento. Esto requiere una pequeña explicación:

Cada instancia-en renderer este ejemplo, una instancia de esta `EllipseViewRenderer` clase-mantiene una única instancia de un objeto nativo, en este ejemplo una Elipse.

Sin embargo, la infraestructura de prestación tiene una facilidad tanto para adjuntar una instancia de procesador a un elemento `Xamarin.Forms` y para separarlo y adjuntar otro elemento `Xamarin.Forms` con el mismo procesador. Tal vez `Xamarin.Forms` tiene que volver a crear el elemento o sustituir por otro elemento para el que ya asociado con el procesador.

Los cambios de este tipo son comunicados al procesador con llamadas a `OnElementChanged`. Los `ElementChangedEventArgs` argumento incluye dos propiedades, `OldElement` y `ElementNuevo`, ambos

del tipo indicado en el argumento genérico para `ElementChangedEventArgs`, en este caso el-
ipseView. En muchos casos, usted no tiene que preocuparse por diferentes elementos Xamarin.Forms se une y se separa de una sola
 instancia de render. Pero en algunos casos es posible que desee utilizar la oportunidad para limpiar o liberar algunos recursos que
 utiliza su procesador.

En el caso más simple y común, cada instancia procesador recibirá una llamada a `OnElement-
 cambiado` para los Xamarin.Forms vista que utiliza ese procesador. Vamos a usar la llamada a `OnElementChanged`
 para crear el elemento nativo y pasarlo a `SetNativeControl`, como ya hemos visto. Después de esa llamada a `SetNativeControl`, el Controlar
 Propiedad Definido por `ViewRenderer` es el objeto nativo, en este caso el Elipse.

En el momento de llegar a esa llamada `OnElementChanged`, el objeto Xamarin.Forms (en este caso una el-
ipseView) probablemente ya ha sido creado y también podría tener algunas propiedades establecen. (En otras palabras, el elemento puede ser
 inicializado con unos valores de propiedades para el momento se requiere que el procesador para mostrar el elemento.) Sin embargo, el
 sistema está diseñado de manera que esto no es necesariamente el caso. Es posible que una llamada posterior a `OnElementChanged` Indica
 que una `EllipseView` Ha sido creado.

Lo que es importante es la `ElementNuevo` propiedad de los argumentos del evento. Si esa propiedad no es nulo
 (Que es el caso normal), que la propiedad es el elemento Xamarin.Forms, y se debe transferir la configuración de propiedades de ese
 elemento Xamarin.Forms al objeto nativo. Ese es el propósito de la llamada a la `SetColor` método mostrado anteriormente. Vas a ver el
 cuerpo de ese método en breve.

los `ViewRenderer` define una propiedad denominada `Elemento` que establece que el elemento Xamarin.Forms, en este caso una `EllipseView`.
 Si la llamada más reciente a `OnElementChanged` que figura un no nulo
`ElementNuevo` propiedad, entonces `Elemento` es que mismo objeto.

En resumen, estas son las dos propiedades esenciales que se pueden utilizar a través de su clase de procesador:

- Elemento -el elemento Xamarin.Forms, válida si el más reciente `OnElementChanged` llamada tenía un no nulo `ElementNuevo` propiedad.
- Controlar -la vista nativo, o un widget, o un objeto de control, válida después de una llamada a `SetNativeView`.

Como se sabe, las propiedades de los elementos Xamarin.Forms pueden cambiar. Por ejemplo, el `Color` propiedad de `EllipseView` podrían ser
 animada. Si una propiedad como `Color` está respaldado por una propiedad enlazable, cualquier cambio en la propiedad que hace que una `PropertyChanged`
 evento que se disparó.

El procesador también es notificado de que el cambio de propiedad. Cualquier cambio en una propiedad que puede vincularse en un elemento de
 Xamarin.Forms conectado a un procesador también hace una llamada a la virtual protegida `OnElementProp-
 ertyChanged` método en el `ViewRenderer` clase. En este ejemplo particular, cualquier cambio en *alguna* propiedad que puede vincularse en `EllipseView`
 (incluyendo el `Color` propiedad) genera una llamada a `OnElementProper-
 tyChanged`. Su procesador debe pasar por encima de ese método y comprobar que la propiedad ha cambiado:

```
espacio de nombres Xamarin.FormsBook.Platform.WinRT
{
    clase pública EllipseViewRenderer : ViewRenderer < EllipseView , Ellipse >
```

```

{
...
protected void override OnElementPropertyChanged ( objeto remitente,
                                              PropertyChangedEventArgs args)
{
    base .OnElementPropertyChanged (remitente, args);

    Si (Args.PropertyName == EllipseView .ColorProperty.PropertyName)
    {
        SetColor ();
    }
...
}
}

```

Si el Color propiedad ha cambiado, la Nombre de la propiedad propiedad del argumento de evento es "Color", el nombre de texto especificado cuando el EllipseView.ColorProperty se creó la propiedad enlazable. Sin embargo, para evitar la falta de ortografía del nombre, el OnElementPropertyChanged método comprueba el valor de cadena real de la propiedad enlazable. El procesador debe responder mediante la transferencia de esa nueva configuración de la Color propiedad al objeto nativo, en este caso el de Windows Elipse objeto.

Esta SetColor método es llamado desde sólo dos lugares: el OnElementChanged anular y la OnElementPropertyChanged anular. No creo que se puede omitir la llamada en OnElementChanged bajo el supuesto de que la propiedad no ha cambiado antes de la llamada a OnElementChanged. Es muy frecuente que las OnElementChanged se llama después un elemento se ha inicializado con valores de propiedades.

Sin embargo, SetColor puede hacer algunas suposiciones válidas acerca de la existencia del elemento Xamarin.Forms y el control natal: Cuando SetColor se llama a partir OnElementChanged, el control nativo ha sido creado y ElementoNuevo es no- nulo. Esto significa que tanto el Controlar y Elemento propiedades son válidas. Los Elemento la propiedad también es válida cuando OnElementPropertyChanged se llama porque ese es el objeto cuya propiedad ha cambiado.

Esto significa que el SetColor método puede transferir simplemente un color de elemento (el elemento Xamarin.Forms) a Controlar, el objeto nativo. Para evitar conflictos de espacio de nombres, esta SetColor Método califica plenamente todas las referencias a cualquier estructura llamada Color:

```

espacio de nombres Xamarin.FormsBook.Platform.WinRT
{
    clase pública EllipseViewRenderer : ViewRenderer < EllipseView , Ellipse >
    {
        ...
        vacío SetColor ()
        {
            Si (Element.Color == Xamarin.Forms. Color .Defecto)
            {
                Control.Fill = nulo ;
            }
        más
    }
}

```

```
    {
        Xamarin.Forms.Color color = Element.Color,
        global :: Windows.UI.Color WinColor =
            global :: Windows.UI.Color.FromArgb (( byte ) (* Color.A 255),
                ( byte ) (* Color.R 255),
                ( byte ) (* Color.G 255),
                ( byte ) (Color.B * 255)),

        Control.Fill = nuevo SolidColorBrush (WinColor);
    }
}
```

Las ventanas Ellipse objeto tiene una propiedad denominada Llenar de tipo Cepillo. Por defecto, esta propiedad es nulo, y eso es lo que el SetColor método establece que si el Color propiedad de EllipseView es Color.Default. De lo contrario, los Xamarin.Forms Color debe ser convertido a una cuenta de Windows Color, que luego se pasa a la SolidColorBrush constructor. los SolidColorBrush objetos se establece en el Llenar propiedad de Ellipse.

Esa es la versión de Windows, pero cuando llega el momento de crear iOS y Android para extracción de grasas elipseView, usted puede sentir un poco frustrado. También en este caso son las restricciones para el segundo parámetro genérico para ViewRenderer:

- iOS: TNativeView está limitada a UIKit.UIView
 - Androide: TNativeView está limitada a Android.Views.Views
 - ventanas: TNativeElement está limitada a Windows.UI.Xaml.FrameworkElement

Esto significa que para hacer una `EllipseView` renderizador para iOS, se necesita una `UIView` derivado que muestra una elipse. Algo como eso existe? No, no lo hace. Por lo tanto, debe hacer uno mismo. Este es el primer paso para hacer que el procesador de iOS.

Por esa razón, el **Xamarin.FormsBook.Platform.iOS** librería contiene una clase llamada **Eipse-UIView** que deriva de **UIView** con el único fin de preparar una ellipse:

```
utilizando CoreGraphics;  
utilizando UIKit;  
  
espacio de nombres Xamarin.FormsBook.Platform.iOS  
{  
    clase pública EllipseUIView : UIView  
    {  
        UIColor color = UIColor .Claro;  
  
        público EllipseUIView ()  
        {  
            BackgroundColor = UIColor .Claro;  
        }  
    }  
}
```

La clase anula la `OnDraw` método para crear un camino de gráficos de una elipse y luego dibujarlo en el contexto gráfico. El color que utiliza se almacena como un campo y se establece inicialmente en `UIColor.Clear`, que es transparente. Sin embargo, se dará cuenta de una `SetColor` método en la parte inferior. Esto ofrece nuevo color a las llamadas de clase y luego `SetNeedsDisplay`, lo que invalida la superficie de dibujo y genera otra llamada a `OnDraw`.

Observe también que el Color de fondo del UIView se encuentra en el constructor para UIColor.Clear.

Sin esa configuración, la vista tiene un fondo negro en el área no cubierta por la elipse.

Ahora que la `EllipseUIView` existe clase para iOS, las `EllipseViewRenderer` puede escribirse usando

EllipsesUIView como el control natal. Estructuralmente, esta clase es prácticamente idéntico al procesador de Windows:

```
utilizando System.ComponentModel;  
  
utilizando UIKit;  
  
utilizando Xamarin.Forms;  
utilizando Xamarin.Forms.Platform.iOS;  
  
[ montaje : ExportRenderer ( tipo de (Xamarin.FormsBook.Platform. EllipseView ),  
                           tipo de (Xamarin.FormsBook.Platform.iOS. EllipseViewRenderer ))  
  
espacio de nombres Xamarin.FormsBook.Platform.iOS  
{  
    clase pùblica EllipseViewRenderer : ViewRenderer < EllipseView , EllipseView >
```

```

    {
        protegido override void OnElementChanged ( ElementChangedEventArgs < EllipseView > args)
        {
            base .OnElementChanged (args);

            Si (control == null )
            {
                SetNativeControl ( nuevo EllipseUIView ());
            }

            Si (Args.NewElement != null )
            {
                SetColor ();
            }
        }

        protegido override void OnElementPropertyChanged ( objeto remitente,
                                                       PropertyChangedEventArgs args)
        {
            base .OnElementPropertyChanged (remitente, args);

            Si (Args.PropertyName == EllipseView .ColorProperty.PropertyName)
            {
                SetColor ();
            }
        }

        vacio SetColor ()
        {
            Si (Element.Color == Color .Defecto)
            {
                Control.SetColor (Element.Color.ToUIColor ());
            }
            más
            {
                Control.SetColor ( UIColor .Claro);
            }
        }
    }
}

```

Las únicas diferencias reales entre el procesador y la versión de Windows es que el Controlar propiedad está establecida en una instancia de `ColorUIView`, y el cuerpo de la `SetColor` método en la parte inferior es diferente. Ahora se llama el `SetColor` método en el `ColorUIView`. Esta `SetColor` método también es capaz de hacer uso de un método de extensión pública en el `Xamarin.Forms.Platform.iOS` biblioteca llamada `ToUIColor`

para convertir un color a un color `Xamarin.Forms IOS`.

Usted puede haber notado que ni el procesador de Windows ni el procesador de iOS tenían que preocuparse por el tamaño. Como veremos en breve, una `EllipseView` Se puede establecer en una variedad de tamaños, y el tamaño calculado en el sistema `Xamarin.Forms` diseño se convierte en el tamaño del control natal.

Esto, desafortunadamente, resultó *no* de ser el caso con el procesador de Android. El procesador de Android

necesita un poco de lógica de dimensionamiento. Al igual que iOS, Android también le falta un control nativo que trasmite una elipse. Por lo tanto, la **Xamarin.Forms.DrawableView** clase contiene una clase llamada **EllipseDrawableView** que deriva de **View** y dibuja una elipse:

```
utilizando Android.Content;
utilizando Android.Views;
utilizando Android.Graphics.Drawables;
utilizando Android.Graphics.Drawables.Shapes;
utilizando Android.Graphics;

espacio de nombres Xamarin.FormsBook.Platform.Android
{
    clase pública EllipseDrawableView : View
    {
        ShapeDrawable dibujable;

        público EllipseDrawableView ( Contexto contexto): base (contexto)
        {
            = dibujable nuevo ShapeDrawable ( nuevo Forma oval ());
        }

        protected void override OnDraw ( Lona lona)
        {
            base .OnDraw (lona);
            drawable.Draw (lona);
        }

        public void SetColor (Xamarin.Forms. Color color)
        {
            drawable.Paint.SetARGB (( Ent ) (255 * color.A),
                ( Ent ) (255 * color.R),
                ( Ent ) (255 * color.G),
                ( Ent ) (255 * color.B));
            Inicializar();
        }

        public void SetSize ( doble anchura, doble altura)
        {
            flotador pixelsPerDip = Resources.DisplayMetrics.Density;
            drawable.SetBounds (0, 0, ( Ent ) (Width * pixelsPerDip),
                ( Ent ) (Altura * pixelsPerDip));
            Inicializar();
        }
    }
}
```

Estructuralmente, esto es similar a la **EllipseUIView** clase definida para iOS, excepto que el constructor crea una **ShapeDrawable** objeto para una elipse, y el **OnDraw** override la hace.

Esta clase tiene dos métodos para establecer las propiedades de esta elipse. Los **SetColor** método convierte un color **Xamarin.Forms** para establecer el **Pintar** propiedad de la **ShapeDrawable** objeto, y el **SetSize** método convierte un tamaño en unidades independientes del dispositivo a los pixeles de fijación de los límites de la **ShapeDrawable**.

objeto. Ambos SetColor y SetSize concluir con una llamada a Invalidate para invalidar la superficie de dibujo y generar otra llamada a OnDraw.

El procesador de Android hace uso de la EllipseDrawableView clase como su objeto nativo:

```
utilizando System.ComponentModel;  
  
utilizando Xamarin.Forms;  
utilizando Xamarin.Forms.Platform.Android;  
  
[ montaje : ExportRenderer ( tipo de (Xamarin.FormsBook.Platform.EllipseView ),  
                           tipo de (Xamarin.FormsBook.Platform.Android.EllipseViewRenderer )))  
  
espacio de nombres Xamarin.FormsBook.Platform.Android  
{  
    clase publica EllipseViewRenderer : ViewRenderer < EllipseView , EllipseDrawableView >  
    {  
        doble anchura, altura;  
  
        protegido override void OnElementChanged ( ElementChangedEventArgs < EllipseView > args)  
        {  
            base .OnElementChanged (args);  
  
            Si ( == control nulo )  
            {  
                SetNativeControl ( nuevo EllipseDrawableView (Contexto));  
            }  
  
            Si (Args.NewElement! = nulo )  
            {  
                SetColor ();  
                SetSize ();  
            }  
        }  
  
        protegido override void OnElementPropertyChanged ( objeto remitente,  
                                                       PropertyChangedEventArgs args)  
        {  
            base .OnElementPropertyChanged (remitente, args);  
  
            Si (Args.PropertyName == VisualElement .WidthProperty.PropertyName)  
            {  
                width = Element.Width;  
                SetSize ();  
            }  
            else if (Args.PropertyName == VisualElement .HeightProperty.PropertyName)  
            {  
                height = element.height;  
                SetSize ();  
            }  
            else if (Args.PropertyName == EllipseView .ColorProperty.PropertyName)  
            {  
                SetColor ();  
            }  
        }  
    }  
}
```

```

        }

        vacio SetColor ()
        {
            Control.SetColor (Element.Color);
        }

        vacio SetSize ()
        {
            Control.SetSize (anchura, altura);
        }
    }
}

```

Observe que el `OnElementPropertyChanged` método necesita para comprobar si hay cambios tanto en el Anchura y Altura propiedades y guardarlos en los campos por lo que se pueden combinar en una sola Límites el establecimiento de la `SetSize` llamar a `EllipseDrawableView`.

Con todos los procesadores de lugar, es el momento para ver si funciona. los **EllipseDemo** solución también contiene enlaces a los distintos proyectos de la **Xamarin.FormsBook.Platform** solución, y cada uno de los proyectos en **EllipseDemo** contiene una referencia al proyecto de biblioteca correspondiente en **Xamarin.FormsBook.Platform**.

Cada uno de los proyectos en **EllipseDemo** También contiene una llamada a la `Toolkit.Init` método en el proyecto de biblioteca correspondiente. Esto no siempre es necesario. Pero hay que tener en cuenta que los distintos prestadores no se hace referencia directamente por cualquier código en cualquiera de los proyectos, y algunas optimizaciones pueden hacer que el código no estar disponible en tiempo de ejecución. La llamada a `Toolkit.Init` evita eso.

El archivo XAML **EllipseDemo** crea varios `EllipseView` objetos con diferentes colores y tamaños, algunos limitados en tamaño, mientras que otros se les permite llenar su contenedor:

```

<? xml versión = " 1.0 " codificación = " utf-8 " ?>
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    xmlns: Plataforma =
        "clr-espacio de nombres: Xamarin.FormsBook.Platform; montaje = Xamarin.FormsBook.Platform "
    x: Class = " EllipseDemo.EllipseDemoPage " >

    < Cuadrado >
        < Plataforma: EllipseView Color = " Agua " />

        < StackLayout >
            < StackLayout.Padding >
                < OnPlatform x: TypeArguments = " Espesor "
                    iOS = " 0, 20, 0, 0 " />
            </ StackLayout.Padding >

                < Plataforma: EllipseView Color = " rojo "
                    WidthRequest = " 40 "
                    HeightRequest = " 80 "
                    HorizontalOptions = " Centrar " />

                < Plataforma: EllipseView Color = " Verde "

```

```
WidthRequest = " 160 "
HeightRequest = " 80 "
HorizontalOptions = " comienzo " />

< Plataforma: EllipseView Color = " Azul "
  WidthRequest = " 160 "
  HeightRequest = " 80 "
  HorizontalOptions = " Fin " />

< Plataforma: EllipseView Color = "# 80FF0000 "
  HorizontalOptions = " Centrar " />

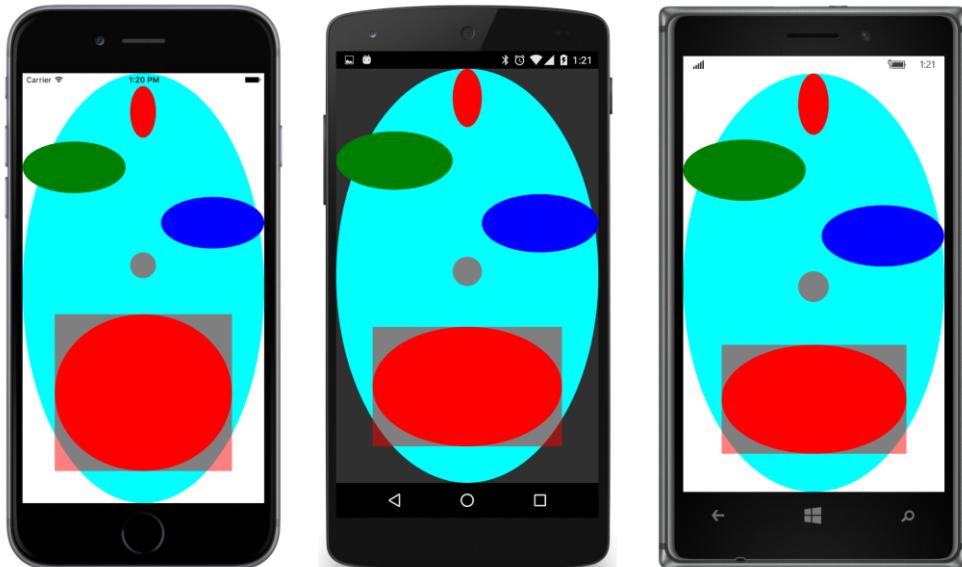
< ContentView Relleno = " 50 "
  VerticalOptions = " FillAndExpand " />

< Plataforma: EllipseView Color = " rojo "
  Color de fondo = "# 80FF0000 " />

</ ContentView >
</ StackLayout >
</ Cuadrícula >
</ Página de contenido >
```

Toma nota en particular de la penúltima EllipseView que se da un color rojo-medio opaco. En contra de Agua de la elipse grande llenado de la página, esto debe rendir como gris medio.

El último EllipseView se da una Color de fondo ajuste de la mitad-opaco de color rojo. De nuevo, esto se debe hacer tan gris contra la gran Agua elipse, sino como una luz roja sobre un fondo blanco y rojo oscuro contra un fondo negro. Aquí están:



Una vez que tenga una EllipseView, por supuesto, usted quiere escribir un programa de rebote de bola. los **Pelota que rebota** solución incluye enlaces a todos los proyectos en el **Xamarin.FormsBook.Platform** solución, y todos los proyectos tienen aplicación referencias a los correspondientes proyectos de biblioteca. los **Pelota que rebota** PCL tiene también una referencia a la **Xamarin.FormsBook.Toolkit** biblioteca una estructura llamada vector2, un vector de dos dimensiones.

El archivo XAML posiciona una EllipseView en el centro de la página:

```
< Pagina de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    xmlns: Plataforma =
        "clr-espacio de nombres: Xamarin.FormsBook.Platform; montaje = Xamarin.FormsBook.Platform "
    x: Class = " BouncingBall.BouncingBallPage " >

    < Plataforma: EllipseView x: Nombre = " pelota "
        WidthRequest = " 100 "
        HeightRequest = " 100 "
        HorizontalOptions = " Centrar "
        VerticalOptions = " Centrar " />

</ Pagina de contenido >
```

El archivo de código subyacente se pone en marcha dos animaciones que se ejecutan "para siempre." La primera animación se define en el constructor y anima el Color propiedad de la pelota que rebota para llevarlo a través de los colores del arco iris cada 10 segundos.

La segunda animación rebota el balón en los cuatro "paredes" de la pantalla. Para cada ciclo a través de la mientras bucle, el código primero determina qué pared que llegará a primera y la distancia a la pared en unidades independiente del dispositivo. El nuevo cálculo de centrar hacia el final de la mientras bucle es la posición de la bola cuando golpea una pared. El nuevo cálculo de vector determina un vector de desviación basado en un vector existente y un vector que es perpendicular a la superficie que se está golpeando (llamado *normal* vector):

```
pública clase parcial BouncingBallPage : Pagina de contenido
{
    pública BouncingBallPage ()
    {
        InitializeComponent ();

        // animación en color: circula por cada 10 segundos de arco iris.
        nuevo Animación (Devolución de llamada: v => ball.Color Color .FromHsla (v, 1, 0,5),
            empezar: 0,
            final: 1
        ) . Commit (propietario: esta ,
            nombre: "ColorAnimation",
            longitud: 10000,
            repetir: () => cierto );
    }

    BounceAnimationLoop ();
}

vacío asíncrono BounceAnimationLoop ()
```

```

{
    // Espera a que las dimensiones son buenas.
    mientras (== Anchura -1 && Altura == -1)
    {
        esperar Tarea .Delay (100);
    }

    // Inicializar puntos y vectores.
    Punto centro = nuevo Punto ();
    Aleatorio rand = nuevo Aleatorio ();
    Vector2 vector = nuevo Vector2 (rand.NextDouble (), rand.NextDouble ());
    vector = vector.Normalized;

    Vector2 [] = {paredes nuevo Vector2 (1, 0), nuevo Vector2 (0, 1),  

                  nuevo Vector2 (-1, 0), nuevo Vector2 (0, -1)};  

                  // izquierda, arriba  

                  // boton derecho

    mientras ( cierto )
    {
        // Las ubicaciones de los cuatro "paredes" (teniendo tamaño de la bola en cuenta).
        doble derecha = Ancho / 2 - ball.Width / 2;
        doble izquierda = -justo;
        doble abajo = Altura / 2 - ball.Height / 2;
        doble TOP = -fondo;

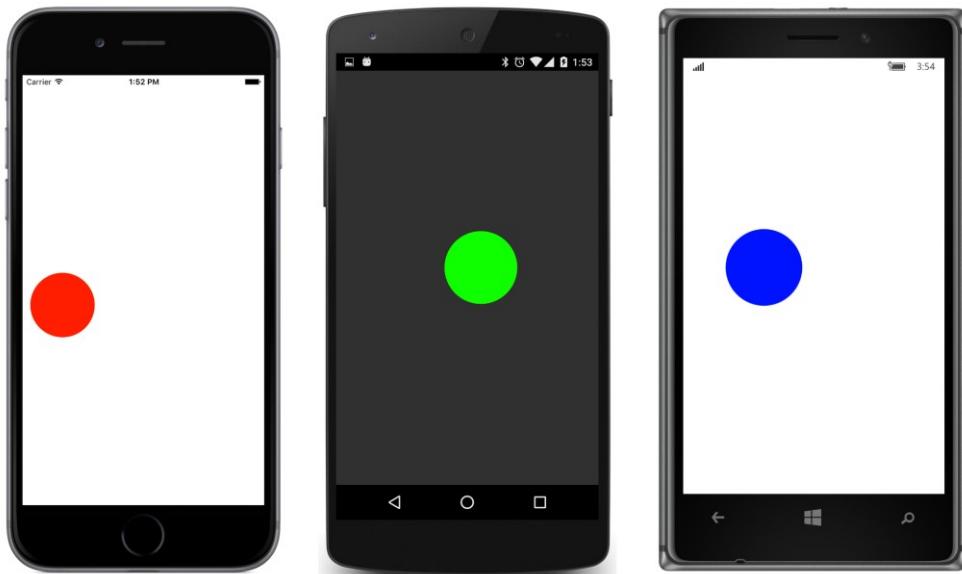
        // Buscar el número de pasos hasta una pared es golpeado.
        doble nX = Mates .Abs (((vector.X> 0 derecha: izquierda) - center.X) / vector.X?);
        doble nY = Mates .Abs (((vector.Y> 0 inferior: superior) - center.Y) / vector.Y);
        doble n = Mates .min (nX, NY);

        // Encuentra la pared que está siendo golpeado.
        Vector2 pared = paredes [nX <nY? (Vector.X> 0 2: 0?): (Vector.Y> 0 3: 1)];

        // Nuevo centro y el vector después de la animación.
        centro += n * vector;
        vector -= 2 * Vector2.DotProduct (vector, pared) * pared;

        // Animate a los 3 mseg por unidad.
        esperar ball.TranslateTo (center.X, center.Y, ( uint ) (3 * n));
    }
}
}
}

Por supuesto, una fotografía todavía no puede captar la emocionante acción de la animación:
```



Extracción de grasas y eventos

La mayoría de los elementos Xamarin.Forms son interactivos. Responden a la entrada del usuario por el disparo de eventos. Si implementa un evento en su elemento Xamarin.Forms costumbre, es probable que también necesite definir un controlador de eventos en los procesadores para el evento correspondiente que los fuegos de control nativas. Esta sección le mostrará cómo.

Los StepSlider elemento fue inspirado por un problema con la implementación de la Xamarin.Forms de Windows deslizador elemento. Por defecto, los Xamarin.Forms deslizador cuando se ejecuta en las plataformas de Windows tiene sólo 10 pasos de 0 a 1, por lo que sólo es capaz de Valor valores de 0, 0,1, 0,2, y así sucesivamente hasta 1,0.

Al igual que los regulares Xamarin.Forms deslizador, el StepSlider elemento tiene Mínimo máximo, y Valor propiedades, sino que también define una Paso propiedad para especificar el número de pasos entre Mínimo y Máximo. Por ejemplo, si Mínimo está ajustado a 5, Máximo está ajustado en 10, y Paso está ajustado en 20, entonces los valores posibles de la Valor propiedad son 5.00, 5.25, 5.50, 5.75, 6.00, y así sucesivamente hasta 10. El número de posibles Valor valores es igual a la Paso valor más 1.

Curiosamente, la aplicación de esta Paso propiedad resultó requerir un enfoque diferente en las tres plataformas, pero el propósito principal de este ejercicio es demostrar cómo implementar eventos.

Aquí está el StepSlider clase en el **Xamarin.FormsBook.Platform** biblioteca. Note la definición de la ValueChanged evento en la parte superior y el despido de ese evento por los cambios en la Valor propiedad. Gran parte de la mayor parte de las definiciones de propiedades enlazables se dedican a la validateValue métodos,

que aseguran que la propiedad se encuentra dentro de los límites permisibles, y la coerceValue métodos, que aseguran que las propiedades son consistentes entre sí:

```
espacio de nombres Xamarin.FormsBook.Platform
{
    clase pública StepSlider : Ver
    {
        evento público Controlador de eventos < ValueChangedEventArgs > ValueChanged;

        sólo lectura estática pública BindableProperty MinimumProperty =
            BindableProperty .Crear(
                "Mínimo",
                tipo de ( doble ),
                tipo de ( StepSlider ),
                0.0,
                validateValue: (obj, min) => ( doble ) Min <(( StepSlider ) Obj) .Maximum,
                coerceValue: (obj, min) =>
                {
                    StepSlider stepSlider = ( StepSlider ) Obj;
                    stepSlider.Value = stepSlider.Coerce (stepSlider.Value,
                        ( doble ) Min,
                        stepSlider.Maximum);
                    regreso min;
                });
        sólo lectura estática pública BindableProperty MaximumProperty =
            BindableProperty .Crear(
                "Máximo",
                tipo de ( doble ),
                tipo de ( StepSlider ),
                100.0,
                validateValue: (obj, max) => ( doble ) Máx > (( StepSlider ) Obj) .Minimum,
                coerceValue: (obj, max) =>
                {
                    StepSlider stepSlider = ( StepSlider ) Obj;
                    stepSlider.Value = stepSlider.Coerce (stepSlider.Value,
                        stepSlider.Minimum,
                        ( doble ) Max);
                    regreso max;
                });
        sólo lectura estática pública BindableProperty StepsProperty =
            BindableProperty .Crear(
                "Pasos",
                tipo de ( Ent ),
                tipo de ( StepSlider ),
                100,
                validateValue: (obj, pasos) => ( Ent ) pasos> 1);

        sólo lectura estética pública BindableProperty ValueProperty =
            BindableProperty .Crear(
                "Valor",
                tipo de ( doble ),
                tipo de ( StepSlider ),
```

```

    0.0,
    BindingMode.TwoWay,
    coerceValue: (obj, valor) =>
    {
        StepSlider stepSlider = (StepSlider) Obj;
        regreso stepSlider.Coerce ((doble) valor,
                                    stepSlider.Minimum,
                                    stepSlider.Maximum);
    },
    PropertyChanged: (obj, oldValue, newValue) =>
    {
        StepSlider stepSlider = (StepSlider) Obj;
        Controlador de eventos < ValueChangedEventArgs > Handler = stepSlider.ValueChanged;
        Si (ManejadorI == nulo)
            Handler (obj, nuevo ValueChangedEventArgs ((doble) valor antiguo,
                                                (doble) nuevo valor));
    });
}

pública doble Mínimo
{
    conjunto (EstablecerValor (MinimumProperty, valor));
    obtener { regreso (doble) GetValue (MinimumProperty); }
}

pública doble Máximo
{
    conjunto (EstablecerValor (MaximumProperty, valor));
    obtener { regreso (doble) GetValue (MaximumProperty); }
}

pública int Pasos
{
    conjunto (EstablecerValor (StepsProperty, valor));
    obtener { regreso (Ent) GetValue (StepsProperty); }
}

pública doble Valor
{
    conjunto (EstablecerValor (ValueProperty, valor));
    obtener { regreso (doble) GetValue (ValueProperty); }
}

doble Obligar( doble valor, doble min, doble max)
{
    regreso Mates.Máximo_minimo, Mates.min (valor, max);
}
}
}

```

los StepSlider fungen de clase las ValueChanged cuando la propiedad Valor propiedad cambia, pero no hay nada en esta clase que cambia el Valor propiedad cuando el usuario manipula el procesador de plataforma para StepSlider. Lo que queda a la clase de procesador.

Una vez más, vamos a primer vistazo a la aplicación de Windows de StepSliderRenderer en el **Xamarin.FormsBook.Platform.WinRT** proyecto compartido, porque es un poco más sencillo. El procesador utiliza la **Windows.UI.Xaml.Controls.Slider** para el control natal.

Para evitar un choque entre el espacio de nombres de Windows deslizador y los **Xamarin.Forms** deslizador, un utilizando Directiva define la

ganar prefijo para referirse al espacio de nombres de Windows y utiliza ese hacer referencia a la de Windows deslizador:

utilizando System.ComponentModel;

utilizando Xamarin.Forms;

utilizando Gana = Windows.UI.Xaml.Controls;

utilizando Windows.UI.Xaml.Controls.Primitives;

Si WINDOWS_UWP

utilizando Xamarin.Forms.Platform.UWP;

#ELSE

utilizando Xamarin.Forms.Platform.WinRT;

terminara si

```
[ montaje : ExportRenderer ( tipo de (Xamarin.FormsBook.Platform. StepSlider ),
                           tipo de (Xamarin.FormsBook.Platform.WinRT. StepSliderRenderer ))]
```

espacio de nombres Xamarin.FormsBook.Platform.WinRT

{

 clase pública StepSliderRenderer : ViewRenderer < StepSlider , Gana. deslizador >

{

 protegido override void OnElementChanged (ElementChangedEventArgs < StepSlider > args)

{

 base .OnElementChanged (args);

 Si (== control nulo)

{

 SetNativeControl (nuevo Gana. deslizador);

}

 Si (Args.NewElement! = nulo)

{

 SetMinimum ();

 SetMaximum ();

 SetSteps ();

 Valor ajustado();

 Control.ValueChanged + = OnWinSliderValueChanged;

}

 más

{

 Control.ValueChanged - = OnWinSliderValueChanged;

}

}

...

}

}

La gran diferencia entre este procesador y el que usted ha visto anteriormente es que éste establece un controlador de eventos en el ValueChanged caso del nativo de Windows Deslizador. (Verá el controlador de eventos en breve.) Si args.NewElement se convierte nulo, sin embargo, eso significa que ya no es un elemento Xamarin.Forms adjunto al procesador y que el controlador de eventos ya no es necesaria. Por otra parte, pronto verás que el controlador de eventos se refiere a la Elemento propiedad heredada de la ViewRe-

derer clase, y que la propiedad será también nulo Si args.NewElement es nulo.

Por esta razón, OnElementChanged se separa el controlador de eventos cuando args.NewElement se convierte nulo. Del mismo modo, todos los recursos que ha asignado para el procesador deben ser liberados cuando args.NewElement se convierte nulo.

La anulación de la OnElementPropertyChanged método comprueba para los cambios en las cuatro propiedades que StepSlider define:

```
espacio de nombres Xamarin.FormsBook.Platform.WinRT
{
    clase pública StepSliderRenderer : ViewRenderer < StepSlider , Gener. deslizador >
    {
        ...
        protected void override OnElementPropertyChanged ( objeto remitente,
                                                       PropertyChangedEventArgs args)
        {
            base .OnElementPropertyChanged (remitente, args);

            Si (Args.PropertyName == StepSlider .MinimumProperty.PropertyName)
            {
                SetMinimum ();
            }
            else if (Args.PropertyName == StepSlider .MaximumProperty.PropertyName)
            {
                SetMaximum ();
            }
            else if (Args.PropertyName == StepSlider .StepsProperty.PropertyName)
            {
                SetSteps ();
            }
            else if (Args.PropertyName == StepSlider .ValueProperty.PropertyName)
            {
                Valor ajustado();
            }
        }
        ...
    }
}
```

Las ventanas deslizador define Mínimo máximo, y Valor Al igual que las propiedades Xamarin.Forms deslizador y el nuevo StepSlider. Pero no define una Pasos propiedad. En su lugar, se define una Paso-Frecuencia propiedad, que es lo contrario de una Pasos propiedad. Para reproducir el ejemplo anterior (Mínimo ajustado a 5, Máximo ajustado a 10, y Pasos configura a 20), lo haces con StepFrequency a 0.25. La conversión es bastante simple:

```

espacio de nombres Xamarin.FormsBook.Platform.WinRT
{
    clase pública StepSliderRenderer : ViewRenderer < StepSlider , Ganan. deslizador >
    {
        ...
        vacío SetMinimum ()
        {
            Control.Minimum = Element.Minimum;
        }

        vacío SetMaximum ()
        {
            Control.Maximum = Element.Maximum;
        }

        vacío SetSteps ()
        {
            Control.StepFrequency = (Element.Maximum - Element.Minimum) / Element.Steps;
        }

        vacío Valor ajustado()
        {
            Control.Value = Element.Value;
        }
        ...
    }
}

```

Por último, aquí está la ValueChanged controlador para Windows Deslizador. Esto tiene la responsabilidad de establecer el Valor propiedad en el StepSlider, que a su vez dispara su propio ValueChanged evento. Sin embargo, existe un método especial para el establecimiento de un valor de un procesador. Este método, llamado SetValueFromRender, se define por la IElementController interfaz y ejecutado por los Xamarin.Forms Element clase:

```

espacio de nombres Xamarin.FormsBook.Platform.WinRT
{
    clase pública StepSliderRenderer : ViewRenderer < StepSlider , Ganan. deslizador >
    {
        ...
        vacío OnControlValueChanged ( objeto remitente, RangeBaseValueChangedEventArgs args)
        {
            (( IElementController ) Element) .SetValueFromRenderer ( StepSlider .ValueProperty,
                args.NewValue);
        }
    }
}

```

el iOS UISlider tiene MinValue, MaxValue, y Valor propiedades y define una ValueChanged acontecimiento, pero no tiene nada parecido a una Pasos o StepFrequency propiedad. En cambio, el iOS StepSliderRenderer clase de **Xamarin.FormsBook.Platform.iOS** hace un ajuste manual de la Valor antes de llamar a la propiedad SetValueFromRenderer desde el ValueChanged controlador de eventos:

```
utilizando Sistema;
utilizando System.ComponentModel;

utilizando UIKit;

utilizando Xamarin.Forms;
utilizando Xamarin.Forms.Platform.iOS;

[ montaje : ExportRenderer ( tipo de (Xamarin.FormsBook.Platform. StepSlider),
    tipo de (Xamarin.FormsBook.Platform.iOS. StepSliderRenderer ))]

espacio de nombres Xamarin.FormsBook.Platform.iOS
{
    clase pública StepSliderRenderer : ViewRenderer < StepSlider , UISlider >
    {
        En 1 pasos;

        protected void override OnElementChanged ( ElementChangedEventArgs < StepSlider > args)
        {
            base .OnElementChanged (args);

            Si ( == control nulo )
            {
                SetNativeControl ( nuevo UISlider ());
            }

            Si (Args.NewElement!= nulo )
            {
                SetMinimum ();
                SetMaximum ();
                SetSteps ();
                Valor ajustado();

                Control.ValueChanged += OnUISliderValueChanged;
            }
            más
            {
                Control.ValueChanged -= OnUISliderValueChanged;
            }
        }

        protected void override OnElementPropertyChanged ( objeto remitente,
                                                       PropertyChangedEventArgs args)
        {
            base .OnElementPropertyChanged (remitente, args);

            Si (Args.PropertyName == StepSlider .MinimumProperty.PropertyName)
            {
                SetMinimum ();
            }
            else if (Args.PropertyName == StepSlider .MaximumProperty.PropertyName)
            {
                SetMaximum ();
            }
        }
    }
}
```

```

else if (Args.PropertyName == StepSlider .StepsProperty.PropertyName)
{
    SetSteps ();
}

else if (Args.PropertyName == StepSlider .ValueProperty.PropertyName)
{
    Valor ajustado();
}

}

vacío SetMinimum ()
{
    Control.MinValue = ( flotador ) Element.Minimum;
}

vacío SetMaximum ()
{
    Control.MaxValue = ( flotador ) Element.Maximum;
}

vacío SetSteps ()
{
    pasos = Element.Steps;
}

vacío Valor ajustado()
{
    Control.Value = ( flotador ) Element.Value;
}

vacío OnUISliderValueChanged ( objeto remitente, EventArgs args )
{
    doble de incremento = (Element.Maximum - Element.Minimum) / Element.Steps;
    doble = valor mínimo de la subesta * Mates .Round (Control.Value / incremento);
    (( IElementController ) Element) .SetValueFromRenderer ( StepSlider .ValueProperty, valor );
}
}

```

Curiosamente, el Android Barra de búsqueda widget tiene un equivalente a la Pasos propiedad, pero no hay equivalentes a la Mínimo y Máximo propiedades! ¿Cómo es esto posible? los Barra de búsqueda en realidad define una propiedad entera llamada Max, y el Progreso propiedad de la Barra de búsqueda es siempre un número entero que varía de 0 a Max. Entonces el Max propiedad realmente indica el número de pasos de las Barra de búsqueda puede hacer, y es necesaria una conversión entre el Progreso propiedad de la Barra de búsqueda y el Valor

propiedad de la StepSlider.

Esta conversión se produce en dos lugares: La `Valor ajustado` método convierte a partir de la `Valor` propiedad de la `StepSlider` al `Progreso` propiedad de la Barra de búsqueda, y el `OnProgressChanged` método convierte a partir de la `Progreso` propiedad de la Barra de búsqueda al `Valor` propiedad de la `StepSlider`.

Además, el controlador de eventos es un poco diferente. El SetOnSeekBarChangeListener método acepta un argumento de tipo `IOnSeekBarChangeListener`, que define tres métodos que informan

cambios en el Barra de búsqueda, incluyendo el método OnProgressChanged. El procesador en sí implementa esa interfaz.

Aquí está la completa StepSliderRenderer clase en el Xamarin.FormsBook.Platform.Android

biblioteca:

utilizando System.ComponentModel;

utilizando Android.Widget;

utilizando Xamarin.Forms;

utilizando Xamarin.Forms.Platform.Android;

```
[ montaje : ExportRenderer ( tipo de (Xamarin.FormsBook.Platform. StepSlider ),
                           tipo de (Xamarin.FormsBook.Platform.Android. StepSliderRenderer ))]

espacio de nombres Xamarin.FormsBook.Platform.Android
{
    clase pública StepSliderRenderer : ViewRenderer < StepSlider , Barra de búsqueda >,
                                         Barra de búsqueda . IOnSeekBarChangeListener
    {
        doble mínimo máximo;

        protected void override OnElementChanged ( ElementChangedEventArgs < StepSlider > args)
        {
            base .OnElementChanged (args);

            Si (== control nulo )
            {
                SetNativeControl ( nuevo Barra de búsqueda (Contexto));
            }
            Si (Args.NewElement! = nulo )
            {
                SetMinimum ();
                SetMaximum ();
                SetSteps ();
                Valor ajustado();

                Control.SetOnSeekBarChangeListener ( esta );
            }
            más
            {
                Control.SetOnSeekBarChangeListener ( nulo );
            }
        }

        protected void override OnElementPropertyChanged ( objeto remitente,
                                                       PropertyChangedEventArgs args)
        {
            base .OnElementPropertyChanged (remitente, args);

            Si (Args.PropertyName == StepSlider .MinimumProperty.PropertyName)
            {
                SetMinimum ();
            }
        }
    }
}
```

```
        }

        else if (Args.PropertyName == StepSlider .MaximumProperty.PropertyName)
        {
            SetMaximum ();
        }

        else if (Args.PropertyName == StepSlider .StepsProperty.PropertyName)
        {
            SetSteps ();
        }

        else if (Args.PropertyName == StepSlider .ValueProperty.PropertyName)
        {
            Valor ajustado();
        }
    }

vacio SetMinimum ()
{
    Element.Minimum mínimo =;
}

vacio SetMaximum ()
{
    máximo = Element.Maximum;
}

vacio SetSteps ()
{
    Control.Max = Element.Steps;
}

vacio Valor ajustado()
{
    doble value = Element.Value;
    Control.Progress = (En t) ((Valor - mínimo) / (máximo - mínimo) * Element.Steps);
}

// Implementación de SeekBar.IOnSeekBarChangeListener
public void OnProgressChanged (Barra de búsqueda barra de búsqueda, En t Progreso, bool Del usuario)
{
    doble value = mínimo + (máximo - mínimo) * Control.Progress / Control.Max;
    ((IElementController ) Element).SetValueFromRenderer (StepSlider .ValueProperty, valor);
}

public void OnStartTrackingTouch (Barra de búsqueda barra de búsqueda)
{
}

public void OnStopTrackingTouch (Barra de búsqueda barra de búsqueda)
{
}

}
```

los **StepSliderDemo** solución contiene enlaces a la **Xamarin.FormsBook.Platform** y bibliotecas

las referencias a las bibliotecas correspondientes. El archivo StepSliderDemo.xaml instancia de cinco StepSlider elementos, con enlaces de datos en tres de ellos y un controlador de eventos explícito sobre los otros dos:

```
< Página de contenido xmlns = " http://xamarin.com/schemas/2014/forms "
    xmlns: x = " http://schemas.microsoft.com/winfx/2009/xaml "
    xmlns: Plataforma =
        "clr-espacio de nombres: Xamarin.FormsBook.Platform; montaje = Xamarin.FormsBook.Platform "
    x: Class = " StepSliderDemo.StepSliderDemoPage " >

< StackLayout Relleno = " 10, 0 " >
    < StackLayout.Resources >
        < ResourceDictionary >
            < Estilo Tipo de objetivo = " ContentView " >
                < Setter Propiedad = " VerticalOptions " Valor = " CenterAndExpand " />
            </ Estilo >

            < Estilo Tipo de objetivo = " Etiqueta " >
                < Setter Propiedad = " Tamaño de fuente " Valor = " Grande " />
                < Setter Propiedad = " HorizontalOptions " Valor = " Centrar " />
            </ Estilo >
        </ ResourceDictionary >
    </ StackLayout.Resources >

    < ContentView >
        < StackLayout >
            < Plataforma: StepSlider x: Nombre = " stepSlider1 " />
            < Etiqueta Texto = " {Binding Fuente = {x: stepSlider1 Referencia},
                Path = Valor} " />
        </ StackLayout >
    </ ContentView >

    < ContentView >
        < StackLayout >
            < Plataforma: StepSlider x: Nombre = " stepSlider2 " 
                Mínimo = " 10 "
                Máximo = " 15 "
                Pasos = " 20 "
                ValueChanged = " OnSliderValueChanged " />
            < Etiqueta x: Nombre = " label2 " />
        </ StackLayout >
    </ ContentView >

    < ContentView >
        < StackLayout >
            < Plataforma: StepSlider x: Nombre = " stepSlider3 " 
                Pasos = " 10 " />
            < Etiqueta Texto = " {Binding Fuente = {x: stepSlider3 Referencia},
                Path = Valor} " />
        </ StackLayout >
    </ ContentView >

    < ContentView >
        < StackLayout >
            < Plataforma: StepSlider x: Nombre = " stepSlider4 " 
                Mínimo = " 0 "
                Pasos = " 10 "
                ValueChanged = " OnSliderValueChanged " />
            < Etiqueta x: Nombre = " label3 " />
        </ StackLayout >
    </ ContentView >
```

```

        Máximo = " 1 "
        Pasos = " 100 "
        ValueChanged = " OnSliderValueChanged " />
    < Etiqueta x: Nombre = " Label4 " />
</ StackLayout >
</ ContentView >

< ContentView >
    < StackLayout >
        < Plataforma: StepSlider x: Nombre = " stepSlider5 " 
            Mínimo = " 10 "
            Máximo = " 20 "
            Pasos = " 2 " />
        < Etiqueta Texto = " {Binding Fuente = {x: stepSlider5 Referencia},
            Path = Valor} " />
    </ StackLayout >
    </ ContentView >
</ StackLayout >
</ Página de contenido >

```

El archivo de código subyacente tiene la ValueChanged controlador de eventos:

```

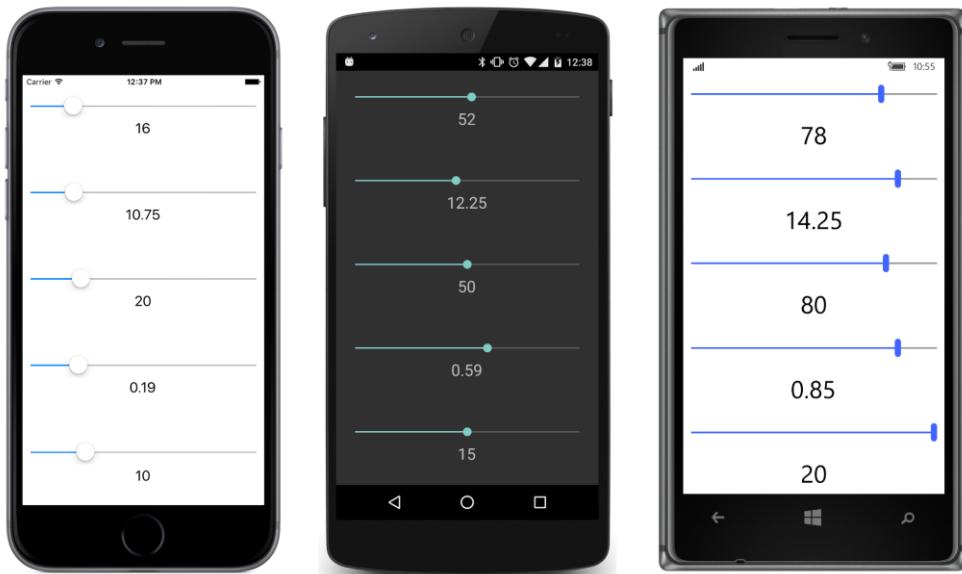
pública clase parcial StepSliderDemoPage : Página de contenido
{
    pública StepSliderDemoPage ()
    {
        InitializeComponent ();
    }

    vacío OnSliderValueChanged ( objeto remitente, ValueChangedEventArgs args )
    {
        StepSlider stepSlider = (StepSlider) remitente;

        Si (StepSlider == stepSlider2)
        {
            Label2.Text = stepSlider2.Value.ToString ();
        }
        else if (StepSlider == stepSlider4)
        {
            label4.Text = stepSlider4.Value.ToString ();
        }
    }
}

```

Usted encontrará que la StepSlider funciona como un Xamarin.Forms normales deslizador excepto que los valores posibles de la StepSlider ahora están bajo control programático:



El primero StepSlider tiene Valor propiedades en incrementos de 1, el segundo en incrementos de 0,25, la tercera en incrementos de 10, la cuarta en incrementos de 0,01, y el quinto en incrementos de 5 con sólo tres configuraciones posibles.

Y ahora se puede ver cómo Xamarin.Forms proporciona las herramientas que le permiten tomar más allá de lo que a primera vista parece ser. Cualquier cosa se puede definir en tres plataformas puede convertirse en algo utilizable en una sola plataforma universal. Con el lenguaje de programación C #, y el poder de Xamarin.Forms y extracción de grasas, puede intervenir no sólo en la programación iOS, Android o programación, o la programación de Windows, pero los tres a la vez con un solo paso, y continúe con el paso hacia el futuro de desarrollo móvil.

Sobre el Autor

Charles Petzold trabaja para Xamarin en el equipo de documentación. Sus primeros años como autónomo se pasó a escribir gran parte de los libros de Microsoft Press acerca de Windows, Windows Phone, y programación .NET, incluyendo seis ediciones de la legendaria *Programación de Windows*, 1988-2012.

Petzold es también el autor de dos libros únicos en los fundamentos matemáticos y filosóficos de la computación digital, [Código: El lenguaje oculto de los Ordenadores, periféricos y software](#) (Microsoft Press, 1999) y *El anotado Turing: Una visita guiada a través del papel histórico de Alan Turing sobre computabilidad y la máquina de Turing* (Wiley, 2008). Vive en la ciudad de Nueva York con su esposa, el escritor Deirdre Sinnott.

Visítenos hoy en



microsoftpressstore.com

- **Cientos de títulos disponibles** - Libros, libros electrónicos, y recursos en línea de expertos del sector
- **envío libre US**
- **libros electrónicos en varios formatos** - leer en el ordenador, tableta, dispositivo móvil o lector electrónico
- **Impresión de libros electrónicos y paquetes Mejor Valor**
- **libro electrónico Oferta de la Semana** - Ahorre hasta un 60% en títulos destacados
- **Boletín de noticias y ofertas especiales** - Sé el primero en conocer los nuevos lanzamientos, ofertas especiales, y más
- **Registre su libro** - Obtener beneficios adicionales



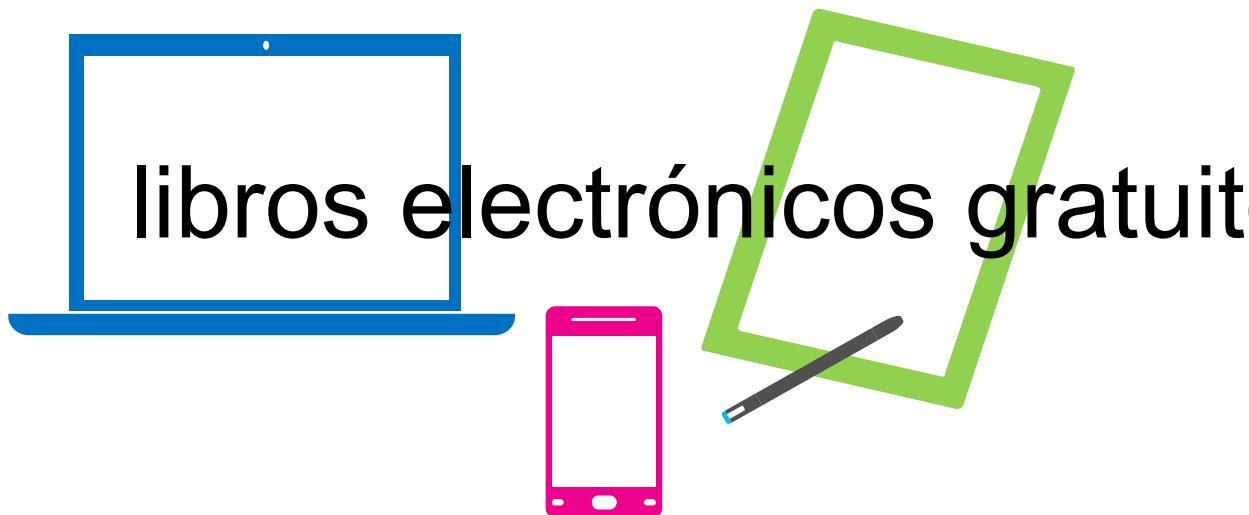
Escuchar acerca de lo primero.



Recibe las últimas noticias de Microsoft Press enviado a su bandeja de entrada.

- libros nuevos y futuros
- Ofertas especiales
- eBooks gratuitos
- Artículos de procedimientos

Inscríbete hoy en MicrosoftPressStore.com/Newsletters



A partir de descripciones técnicas a drilldowns sobre temas especiales, obtener
gratis libros electrónicos de Microsoft Press:

www.microsoftvirtualacademy.com/ebooks

Descargar sus libros electrónicos gratis en PDF, EPUB, y / o Mobi para Kindle formatos.

Busque otros grandes recursos en Microsoft Virtual Academy, donde se puede aprender nuevas habilidades y ayudar a avanzar en su carrera con formación gratuita Microsoft entregado por los expertos.

microsoft Press



Ahora que
usted ha
leído el libro

...

¡Dinos qué piensas!

Fue útil?

¿Se le enseñó lo que quería aprender? ¿Había
margen de mejora?

Háganoslo saber en <http://aka.ms/tellpress>

Su regeneración va directamente al personal de Microsoft Press, y leemos cada
una de sus respuestas. ¡Gracias por adelantado!



Microsoft