Windows 8: Fundamentos de desarrollo

Alex Casquete, Toni Recio

Septiembre 2012

RESUMEN

Windows 8 ya es una realidad. Desde hace unos días tenemos disponible, a través de los diferentes programas de *Partners* o las suscripciones MSDN y TechNet, tanto la versión final del nuevo sistema operativo de Microsoft como de su entorno de desarrollo, Visual Studio 2012. Te invitamos a que conozcas cómo desarrollar aplicaciones para la nueva plataforma, que sin duda se convertirá en la mayor oportunidad que los desarrolladores hemos tenido en los últimos años.

En este artículo introduciremos los conceptos clave del nuevo estilo de interfaz de usuario de Windows 8 así como las novedades que nos presentan la nueva plataforma y los nuevos modelos de desarrollo. Veremos cómo afrontar nuestro primer proyecto y cómo diseñar nuestras aplicaciones para ofrecer la mejor experiencia sacando todo el partido de Windows 8.

Nota: Este artículo se publicó en el número 96 de la revista dotNetManía en septiembre de 2012.

ABSTRACT

Windows 8 is already a reality. For a few days, we have available, through the different Partner programs or MSDN and TechNet subscriptions, both the final version of Microsoft's new operating system and its development environment, Visual Studio 2012. We invite you to learn how to develop applications for the new platform, which will undoubtedly become the most incredible opportunity developers have had in recent years.

This article will introduce the key concepts of the new style of Windows 8 user interface and the innovations presented by the new platform and the new development models. We will see how to face our first project and design our applications to offer the best experience taking full advantage of Windows 8.

Note: This article was published in the issue 96 of September 2012 of the dotNetMania magazine.

1 El renacer del Tablet-PC

Ya con Windows XP, podíamos interactuar con el sistema operativo mediante pantallas táctiles, posteriormente con Vista se trató de incrementar el tamaño de algunos controles para mejorar la experiencia táctil, y con la "dieta" a la que se sometió Windows 7, incluso el asalto a las tabletas se vio hecho realidad. Windows era un sistema operativo que se ejecutaba correctamente en dispositivos móviles, pero no había sido concebido para ello, algo que quedaba de manifiesto en la pobre experiencia que el usuario obtenía en dicho contexto.

¿Pero cómo adaptar el Windows de toda la vida a un nuevo modelo dónde la interfaz gestual sea protagonista, sin romper la compatibilidad con el paradigma anterior? Pues ofreciendo al usuario lo que necesita en cada momento, creando dos experiencias de usuario diferenciadas, la gestual centrada en el uso de pantallas táctiles (Windows RT), y la de escritorio (Windows Desktop), basada en el uso del teclado y ratón. Con Windows 8 podemos viajar en trasporte público con una tableta, interactuar de forma fácil e intuitiva con aplicaciones diseñadas específicamente para ser "tocadas", llegar a casa, y conectar esa misma tableta a un monitor, teclado y ratón, y abrir por ejemplo Visual Studio para desarrollar con nuestro lenguaje de programación favorito. Windows 8 es por tanto, el primer sistema operativo real para Tablet-PC, ya que nos permite conjugar lo mejor de cada mundo en un mismo dispositivo.

Pese al renacimiento del Tablet-PC que impulsa Windows 8, productos como iPad han demostrado que existe un nicho de mercado importante orientado a usuarios, que con una tableta de prestaciones limitadas, ven sus necesidades satisfechas. En ese sentido se comercializará una versión de Windows específica para procesadores ARM dónde sólo dispondremos de Windows RT, sin acceso a nuestras aplicaciones de escritorio de toda la vida, pero a un precio más reducido.

2 Windows 8 UI Style (Codename Metro)

Al diferenciar los dos contextos, RT y escritorio, los equipos de UX de Microsoft han tenido la oportunidad de crear una nueva interfaz visual desde cero, centrada en el lenguaje gestual y en la presentación de contenido, cuyas primeras versiones aparecieron en Windows Phone y Xbox 360 bajo el codename de Metro, y que finalmente en esta nueva entrega ha pasado a denominarse Windows 8 UI Style. Durante su creación sus autores bebieron principalmente de tres fuentes de inspiración:

- El diseño moderno representado por la escuela Bauhaus, el cual se basa en convertir la funcionalidad en algo bello, y que ha hecho famosa la frase del arquitecto Mies van der Rohe de "Menos es más".

- El diseño suizo, también conocido como estilo tipográfico internacional, el cual realza el uso de las tipografías claras, la jerarquización del contenido, y de las iconografías sencillas, al estilo de las indicaciones de los espacios

públicos como los aeropuertos, en busca de un lenguaje universal que muestre la información sin interferencias, del modo más directo posible.

- El diseño del movimiento basado en el leguaje cinematográfico, buscando un impacto emocional a través de animaciones, más que buscar la espectacularidad de efectos visuales que no aportan sentimiento o información.

De este modo, las aplicaciones tradicionales se siguen planteando del mismo modo, pero aquellos desarrollos focalizados en dispositivos móviles deben embeberse en el nuevo modelo; Es en este contexto en el que nace el concepto de Aplicación Windows 8 UI Style.

3 La promiscuidad del mercado móvil

Actualmente uno de los mayores quebraderos de cabeza de los principales actores de la industria del mercado móvil, es la promiscuidad de plataforma, o dicho de otro modo, lo sencillo que es para la mayoría de los usuarios cambiar de plataforma sin demasiados "problemas".

Hasta la fecha, aplicaciones en exclusiva, o llevar los datos de los usuarios a nubes sin soporte multiplataforma, han sido estrategias con pobres resultados, y Windows 8 UI Style llega para ofrecer una interesante vuelta de rosca a la problemática. Por un lado, se plantea una interfaz del sistema operativo novedosa, muy diferente a las clásicas basadas en escritorios con iconos estáticos, y por el otro y todavía más importante, se aplica la misma filosofía de diseño a las aplicaciones en sí. De este modo, el usuario se beneficia de un único proceso de aprendizaje, de una congruencia generalizada del sistema, de forma que renunciar a la plataforma, representa renunciar también a la comodidad y sencillez de uso.

4 Los 5 principios clave de Windows 8 UI Style

En Windows RT nos encontramos con que las aplicaciones ocupan toda la pantalla, a lo sumo pueden compartir una porción de la misma con una segunda aplicación, con lo que decimos adiós a botones de sistema como maximizar o minimizar, haciendo buena la máxima –el contenido antes que el marco-.

El *scroll* se realiza de forma horizontal, y las opciones de sistema se muestran al deslizar los dedos sobre los extremos horizontales de la pantalla. En los extremos verticales aparecen los comandos de la aplicación activa, de modo que la aplicación se centre en el contenido, relegando a un segundo plano las acciones.

Pero para que las aplicaciones cumplan con los estándares de uso, los desarrolladores deben tener en mente 5 principios de diseño que representan la esencia de la nueva filosofía.

Muestra buen hacer (el retorno del artesano) – Y es que no hay mejor manera de fidelizar a un cliente que haciendo bien las cosas, es por ello, que los detalles son primordiales. La jerarquización del contenido, el uso de plantillas de forma

generalizada, tipografías concretas, etc. son claves para proporcionar una experiencia confiable y constante en todo el sistema.

Ser rápido y fluido (*Fast & Fuid*) – El tiempo de respuesta es especialmente crítico en un entorno táctil, con lo que la programación asíncrona de la interfaz de usuario pasa de ser una buena práctica a un requisito.

Auténticamente digital – Sacar provecho de las opciones que nos ofrecen los entornos digitales, como en los iconos vivos, que además de darnos acceso a aplicaciones son partes de la misma, proporcionando información relevante al usuario, o la sincronización de configuraciones a través de la nube.

Hacer más con menos – Las aplicaciones nacen con el objetivo de ser las mejores en un determinado campo. Debemos olvidarnos de los aplicativos que lo hacen todo, pero que no ayudan al usuario en sus necesidades concretas.

Ganar en equipo – El uso de plantillas, o los contratos, que hacen que todas las aplicaciones realicen tareas como buscar y compartir de forma unificada, hacen que los nuevos desarrollos sean más sencillos y consistentes. En definitiva, evitar reinventar la rueda.

5 La nueva plataforma de desarrollo

Desde el punto de vista del desarrollador, la primera gran novedad que nos encontramos es la nueva *API Windows Runtime* (WinRT), diseñada para acceder al sistema y a los servicios de los dispositivos donde se ejecutan las aplicaciones. WinRT nos proporciona cientos de clases que nos permiten desde acceder a la cámara o a los sensores del dispositivo, hasta guardar un archivo o compartir contenido con otras aplicaciones. En la Tabla 1 aparecen las principales API junto con las funcionalidades principales que exponen.

User Interface	Enlace a datos, accesibilidad
Devices	Geolocalización, Sensores, NFC
Media	Reproducción y captura de audio y video, animaciones, efectos visuales
Communication & Data	Ficheros, bases de datos
Fundamental	Gestión de memoria, globalización, seguridad

Tabla 1

Las API de WinRT son nativas, están creadas en C++, y además de estar diseñadas para ofrecer un buen rendimiento, se pueden utilizar fácilmente por todo tipo de desarrolladores. Esto es posible ya que al estar creadas con un modelo unificado de tipos, permite que, mediante proyecciones, se puedan utilizar en varios lenguajes de programación. Esto significa que podemos desarrollar una aplicación Windows 8 UI Style sin tener que aprender un nuevo lenguaje, ni tampoco ceñirnos a uno solo. El lenguaje de programación es solo un medio. A partir de ahora vamos a poder crear aplicaciones nativas para Windows 8 utilizando los lenguajes que ya conocemos: C#, VB.NET, C++ con XAML y JavaScript con HTML 5 y CSS 3, de forma que el modelo de programación con XAML va a permitir a los desarrolladores con conocimientos en

Silverlight o WPF crear aplicaciones de la misma forma que los desarrolladores web lo van a poder hacer con el modelo de programación basado en HTML.

En el caso de las aplicaciones JavaScript, vamos a contar también con la ayuda de *Microsoft Windows Library for JavaScript* (WinJS), una librería que incluye implementaciones de patrones, control de navegación, animaciones y utilidades para facilitar el desarrollo de aplicaciones Windows 8 UI Style con JavaScript.

Como hemos comentado ya, las API de WinRT están diseñadas para ofrecer un buen rendimiento, esto es así porque la mayoría de las operaciones realizadas con WinRT se ejecutan de forma asíncrona. De esta forma se consigue no bloquear en ningún momento la interfaz de usuario. Para trabajar de forma cómoda con este tipo de programación se han dotado a los lenguajes (C#, JavaScript, VB.NET y C++) de patrones que nos facilitan las llamadas a métodos asíncronos. Todo esto lo vamos a ver mejor con nuestro primer ejemplo.

6 Hello Grid

A la hora de comenzar nuestro primer proyecto desde cero tenemos que elegir entre una de las plantillas que nos proporciona Visual Studio o Blend. Saber de qué forma vamos a organizar la información y cuantos niveles de navegación queremos ofrecer en nuestra aplicación, nos ayudará a elegir la plantilla idónea para nuestro proyecto.

Para los proyectos en JavaScript, Visual Studio nos ofrece cinco plantillas de proyecto: vacía (*blank*), navegación (*navigation*), dividida (*split*) y cuadrícula (*grid*). Para los proyectos XAML solo contamos con tres de estas plantillas: vacía, dividida y cuadrícula. La plantilla vacía nos proporciona un proyecto sin ningún tipo de contenido ni funcionalidad. La plantilla de navegación nos proporciona la funcionalidad básica para navegar entre páginas mediante el modelo de navegación de página única y haciendo uso del control de navegación que está implementado en el fichero navigator.js. La plantilla de aplicación dividida y de cuadrícula, además del modelo de navegación, nos ofrece varias vistas para un modelo de datos con datos de ejemplo estáticos.

El uso de estas plantillas es la mejor forma de comenzar una aplicación (recordemos, trabajar en equipo) ya que nos proporciona, además de la navegación y el modelo de datos, la gestión para los diferentes estados de vistas (completa, acoplada o rellena).

De todas las plantillas que tenemos disponibles nos vamos a centrar en la plantilla de cuadrícula, que es la más completa, la que nos puede servir para un mayor número de escenarios y representa perfectamente los principios de diseño anteriormente comentados. Esta plantilla contiene tres vistas de la misma información: la vista principal muestra una lista de grupos. Al seleccionar un grupo se muestra el detalle con la lista de elementos que pertenecen a ese grupo colocados a la derecha. Y al pulsar sobre un elemento, ya sea desde la vista de grupo o desde la vista principal, se abre la página de detalle del elemento.

Para mostrar el funcionamiento de esta plantilla vamos a crear una aplicación que muestre todas las portadas de DotNetMania agrupadas por años y en el detalle del elemento nos aparezca el editorial correspondiente a ese número. Para comenzar,

tenemos que crear un proyecto (ya sea utilizando JavaScript o XAML) y asignarle el nombre DotNetMania. Una vez creado, podremos ejecutar la aplicación y navegar por las distintas páginas que aparecen en la Imagen 1.

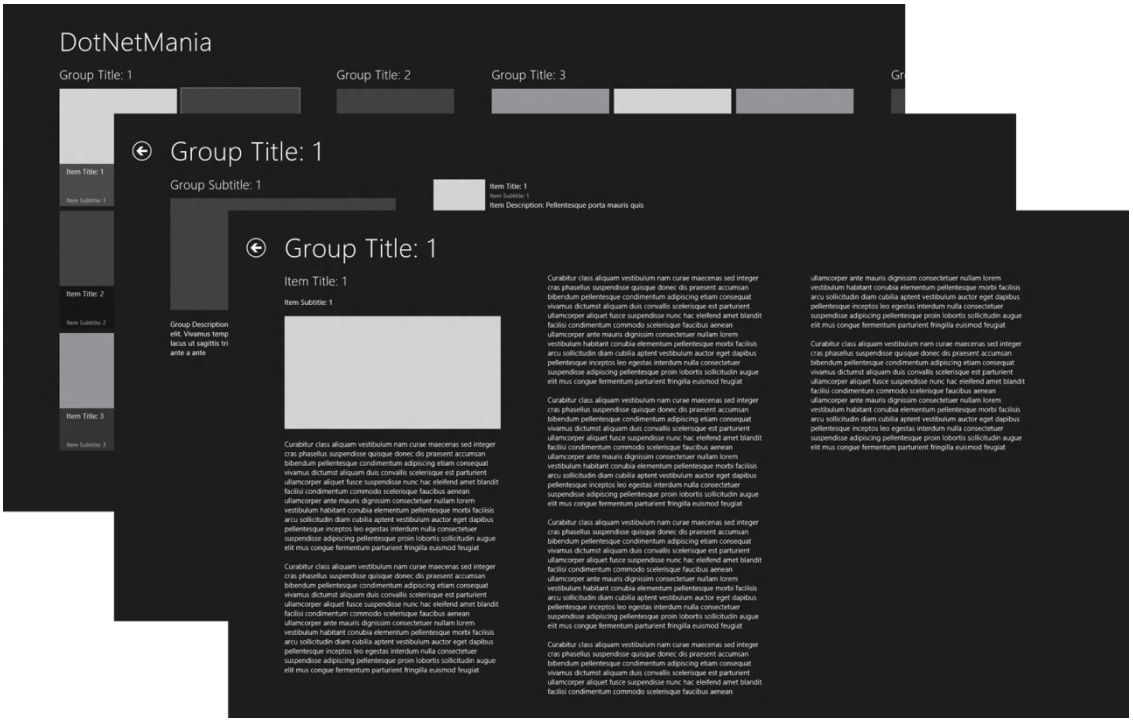
El primer cambio que vamos a realizar es aplicar el estilo del tema alternativo. Por defecto, Visual Studio crea los proyectos con el tema oscuro. Si queremos utilizar el claro tenemos que cambiar las referencias de las páginas HTML a la hoja de estilo **ui-dark.css** por **ui-light.css**. Esto en aplicaciones JavaScript, con XAML el cambio de tema se realiza utilizando la propiedad **RequestedTheme** del objeto **Application** (Código 1). Este simple código nos pone de manifiesto las diferencias entre los dos modelos de programación.

```
JavaScript

<link href="//Microsoft.WinJS.1.0.RC/css/ui-dark.css" rel="stylesheet" />

C#
<Application
    x:Class="DotNetMania.App"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:DotNetMania"
    RequestedTheme="Light">
```

Código 1Definición de estilo claro en HTML y XAML



7 Cambiando el origen de datos

La plantilla de proyecto de cuadrícula crea el fichero **data.js** (en aplicaciones JS) y la clase **SampleDataSource** (en aplicaciones XAML) que expone los datos al resto de aplicación. Estos ficheros contienen por defecto datos estáticos, algo que es poco usual, ya que normalmente vamos a tener que obtener los datos de un origen externo,

como por ejemplo, un servicio web. Vamos a modificar el código para que obtenga toda la información de un servicio WCF que devuelve datos en formato JSON.

En JS, la forma de llamar a un servicio es mediante una llamada a la función WinJS.xhr. Esta función es un contenedor para **XMLHttpRequest** que nos permite de descargar contenido web de una forma sencilla. En el Código 2 se muestra la llamada a un servicio y la forma de añadir los elementos a la lista mediante JavaScript. En C#, sin embargo, el cambio no va a ser tan sencillo, ya que tenemos que cambiar el modelo de datos.

```
WinJS.xhr({ url: "http://techdencias.azurewebsites.net/mainservice.svc" })
    .then(function (xhr) {
        var items = JSON.parse(xhr.responseText);
        items.forEach(function (item) {
            list.push(item);
        });
    });
```

Código 2 Llamada a un servicio web utilizando la función xhr de WinJS

En la clase **SampleDataSource** creamos un método que nos realice la petición al servicio web. Además tenemos que crear el código que procese los datos JSON y cree nuevas instancias de las clases **SampleDataItem** y **SampleDataGroup**. En el ejemplo se está haciendo en el método **ConvertJSONtoObjects**. Estos ejemplos están planteados para realizar el mínimo cambio al código generado por las plantillas, pero sobra decir que en este caso deberíamos crearnos nuestra propia capa de datos y no utilizar las clases **SampleDataItem** y **SampleDataSource**. En el Código 3 se muestra como realizar la llamada a un servicio desde C#. Hemos que tener en cuenta que las páginas de la plantilla cuadrícula están pensadas para trabajar con una determinada estructura de datos. Cuanto más modifiquemos esa estructura, más tendremos que modificar el código para que funcione. Por ejemplo, en nuestros datos la estructura no contiene el campo subtitulo lo que nos provoca que tengamos que modificar las páginas para que no muestren este campo.

```
public async void LoadRemoteDataAsync()
{
    var client = new HttpClient();
    client.MaxResponseContentBufferSize = 1024 * 1024;
    var response = await client.GetAsync(new
Uri("http://techdencias.azurewebsites.net/mainservice.svc"));
    var result = await response.Content.ReadAsStringAsync();
    // Parse JSON y crea objetos
    var data = JsonArray.Parse(result.Substring(1, result.Length - 1));
    ConvertJSONtoObjects(data);
}
```

Código 3 Llamada a un servicio web mediante C#

Estos dos ejemplos nos sirven además para mostrar cómo se ha simplificado la llamada a métodos asíncronos. En el caso de JavaScript está resuelto utilizando el patrón **Promise** y en C# utilizando las nuevas palabras clave **async** y **await**. En JavaScript los métodos asíncronos devuelven un objeto Promise que consiste básicamente en un objeto que tiene el método **then** y al que le pasamos tres funciones (aunque sólo la primera es obligatoria) una función para llamar cuando la **Promise** se completa correctamente, una función para llamar cuando se completa con un error y una función para informar sobre el progreso. En el caso de C#, el modificador **async** del método indica que ese método es asíncrono y el operador **await** aplicado

a la tarea indica que el método debe suspender la ejecución hasta que se complete la tarea.

Por último, y para que la página nos muestre un aspecto similar al que aparece en la Imagen 2, tendremos que realizar cambios en las hojas de estilo y en las plantillas. Por ejemplo, para modificar el ancho de las imágenes de la vista principal tenemos que modificar la hoja de estilo **groupedItems.css** y en XAML tenemos que modificar el **DataTemplate** que se aplica a los elementos y que está definido en el fichero **StandardStyles.xaml** (Código 4).

CSS

```
.groupeditemspage .groupeditemslist .win-item {
    -ms-grid-columns: 1fr;
    -ms-grid-rows: 1fr 90px;
    display: -ms-grid;
    height: 110px;
    width: 47px;
}
```

XAML

```
<DataTemplate x:Key="Standard250x250ItemTemplate">
    <Grid HorizontalAlignment="Left" Width="110" Height="147">
        <Border Background="{StaticResource
ListViewItemPlaceholderBackgroundThemeBrush}">
            <Image Source="{Binding Image}" Stretch="UniformToFill"/>
        </Border>
    </Grid>
</DataTemplate>
```

Código 4 Cambios en la hoja de estilo CSS y XAML para cambiar el tamaño de los elementos

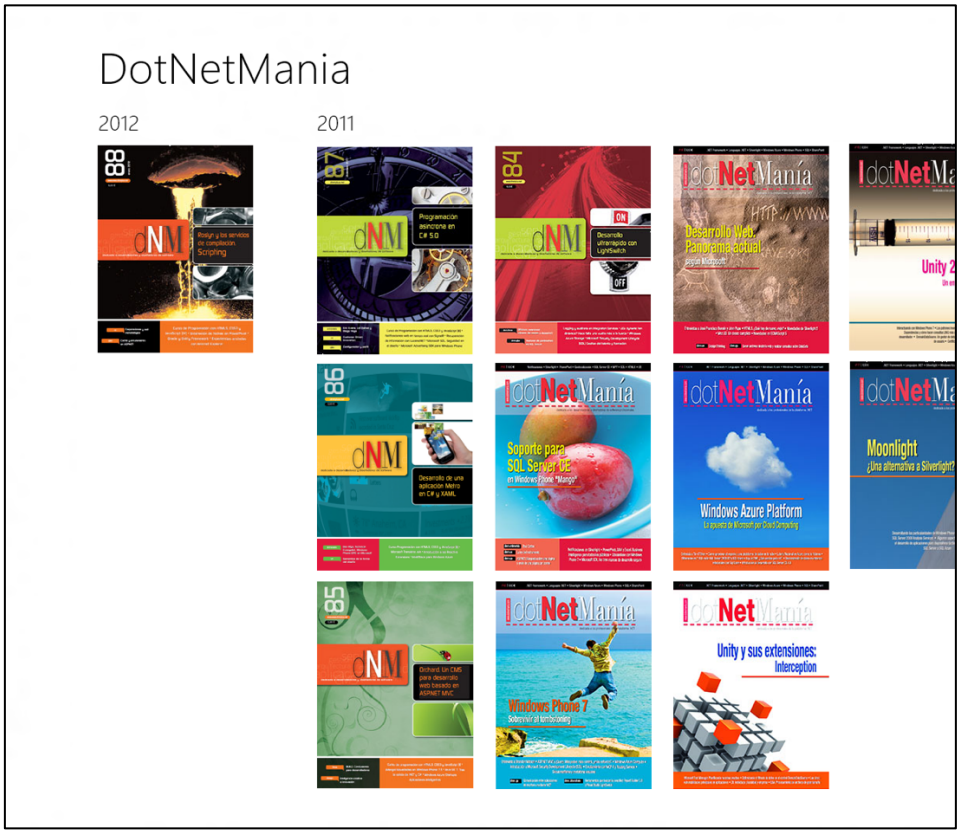


Imagen 2 Aspecto de la vista principal después de los cambios de estilo

8 Ciclo de vida del proceso

En Windows 8 se introduce un nuevo modelo de ciclo de vida de las aplicaciones en el que el sistema suspende de forma automática las aplicaciones que no están en primer plano. Esto significa que una aplicación que está en primer plano tiene asignados todos los recursos disponibles del sistema.

Una aplicación se suspende cuando el usuario cambia a otra aplicación y se reanuda cuando se vuelve a ella. Una aplicación suspendida permanece en memoria, pero no tiene ningún impacto en el procesador ni en el tráfico de red ya que todos los hilos de ejecución se paran. Es un estado similar al que conseguimos pulsando el botón de pausa en Visual Studio. Cuando una aplicación es reanudada todos los hilos continúan con la ejecución y las variables se restauran al mismo estado que se quedaron cuando se suspendió. La experiencia de reanudar una aplicación es instantánea ya que todo el código de la aplicación está en memoria. Además, y según las necesidades del sistema, el propio sistema operativo puede finalizar las aplicaciones que ya estén suspendidas. Esto se hace para ofrecer siempre la mejor experiencia de usuario, sin importar el número de aplicaciones abiertas, y para reducir al máximo el consumo de batería y recursos. Esto, por contra, requiere que nuestras aplicaciones estén diseñadas para restaurar su estado para dar la sensación de que la aplicación nunca ha sido finalizada. El usuario debe ver que todo está como cuando cambió de aplicación, aunque en realidad se haya iniciado de nuevo.

El cambio de estado a suspendida viene precedido por una notificación del sistema, pero la finalización de una aplicación se produce sin que haya ningún tipo de notificación. Por lo tanto la aplicación debe guardar la información cuando es suspendida para poder restaurarla si es finalizada. En las plantillas existe el código mínimo necesario para gestionar el ciclo de vida de la aplicación. Y además, la forma de persistir este estado cambia según utilicemos JavaScript o C#.

En el fichero **default.js** se incluye el código para controlar el evento **oncheckpoint**. Este evento se dispara cuando la aplicación va a ser suspendida y es el momento que tenemos que aprovechar para guardar el estado de la sesión. La plantilla genera el código para guardar el histórico de navegación (Código 5). En este código se está haciendo uso del objeto **sessionState** de WinJS que se serializa a disco durante la suspensión de la aplicación y se recupera cuando la aplicación se activa de nuevo.

```
app.oncheckpoint = function (args) {
    app.sessionState.history = nav.history;
};
```

Código 5 Almacenar estado de la aplicación con JavaScript

En el caso de las aplicaciones XAML tenemos que hacer uso de la clase **SuspensionManager** que se añade a nuestro proyecto cuando utilizamos las plantillas de Visual Studio. Esta clase serializa los datos del estado de la página y lo escribe en un archivo XML en el almacenamiento local de la aplicación. En el archivo App.xaml.cs está el controlador del evento **Suspending** que se llama cuando Windows notifica que la aplicación va a ser suspendida. Al dispararse, se llama al método **SaveAsync** del objeto **SuspensionManager.** Este método guarda el estado de navegación de todos los **Frames** registrados y de cada una de las páginas.

App.xaml.cs

```
private async void OnSuspending(object sender, SuspendingEventArgs e)
{
    var deferral = e.SuspendingOperation.GetDeferral();
    await SuspensionManager.SaveAsync();
    deferral.Complete();
}
```

ItemDetailPage.xaml.cs

```
protected override void SaveState(Dictionary<String, Object> pageState)
{
    var selectedItem = (SampleDataItem)this.flipView.SelectedItem;
    pageState["SelectedItem"] = selectedItem.UniqueId;
}
```

Código 6 Almacenar estado de la aplicación con C#

Observemos ahora como se realiza la restauración del estado de aplicación en el caso de que nuestra aplicación se reactive después de la suspensión. En el controlador del evento **onactivated** del fichero **defaul.js** se comprueba si la aplicación se ha cargado desde suspensión y se restaura el estado de navegación (Código 7). En C# se realiza la misma operación, pero en el método **OnLaunched** de la clase App y en el método **LoadState** de cada una de las páginas (Código 8).

```
app.onactivated = function (args) {
    if (args.detail.kind === activation.ActivationKind.launch) {
        if (args.detail.previousExecutionState !==
activation.ApplicationExecutionState..terminated) {
            // Initialize your application here.
        } else {
            // Restore application state here.
        }

        if (app.sessionState.history) {
            nav.history = app.sessionState.history;
        }
        args.setPromise(WinJS.UI.processAll().then(function () {
            if (nav.location) {
                nav.history.current.initialPlaceholder = true;
                return nav.navigate(nav.location, nav.state);
            } else {
                return nav.navigate(Application.navigator.home);
            }
        }));
    }
});
```

Código 7 Recuperar estado de la aplicación con JavaScript

App.xaml.cs

```
protected override async void OnLaunched(LaunchActivatedEventArgs args)
{
    if (args.PreviousExecutionState == ApplicationExecutionState.Running)
    {
        Window.Current.Activate();
        return;
    }

    var rootFrame = new Frame();
    SuspensionManager.RegisterFrame(rootFrame, "AppFrame");

    if (args.PreviousExecutionState == ApplicationExecutionState.Terminated)
    {
        await SuspensionManager.RestoreAsync();
    }

    if (rootFrame.Content == null)
    {
```

```
        if (!rootFrame.Navigate(typeof(GroupedItemsPage), "AllGroups"))
        {
            throw new Exception("Failed to create initial page");
        }
    }

    Window.Current.Content = rootFrame;
    Window.Current.Activate();
}
```

ItemDetailPage.xaml.cs

```
protected override void LoadState(Object navigationParameter, Dictionary<String,
Object> pageState)
{
    if (pageState != null && pageState.ContainsKey("SelectedItem"))
    {
        navigationParameter = pageState["SelectedItem"];
    }
    …
}
```

Código 8 Recuperar estado de la aplicación con C#

En estos eventos se está comprobando el valor de la propiedad **PreviousExecutionState**. Debemos tener en cuenta que no debemos restaurar el estado de la aplicación si esta finalizó a causa de un error (**NotRunning**) o el usuario cerró la aplicación (**ClosedByUser**). Únicamente tenemos que restaurar el estado si el valor de **PreviousExecutionState** es **Terminated.**

Para finalizar este espacio dedicado al ciclo de vida quedaría comentar un tema importante, ya que según lo contado hasta ahora parece que no podamos ejecutar procesos en segundo plano. Esto no es así, Windows 8 permite ejecutar código mientras nuestra aplicación está suspendida mediante las tareas en segundo plano. El entorno de ejecución de estas tareas es un entorno restringido y solo recibe una cantidad de tiempo de CPU. Por lo tanto, las tareas de segundo plano se deben utilizar para realizar pequeñas tareas que no tengan interacción con el usuario.

9 Integrada con Windows 8

Otra de las novedades que tenemos en Windows 8 es la barra de accesos. Esta barra es la que aparece cuando ubicamos el cursor en la esquina superior derecha, deslizamos en el margen derecho de la pantalla o pulsamos Windows+C y que nos proporciona una forma unificada de acceder a funcionalidades comunes, como realizar una búsqueda o compartir información, en todas las aplicaciones de Windows 8.

Si, por ejemplo, queremos hacer una búsqueda en una aplicación, simplemente tenemos que seleccionar el acceso de búsqueda e introducir el término dentro del cuadro de búsqueda. Mientras escribimos se nos muestran una serie de sugerencias que nos llevan al detalle de esa información y si, por el contrario, pulsamos el botón buscar veremos la vista de resultados de la aplicación. Este comportamiento es el que nos vamos a encontrar en todas las aplicaciones. Pero además, este sistema nos permite que se puedan realizar búsquedas en aplicaciones incluso si no están en ejecución. En la parte inferior del cuadro de búsqueda aparecen todas las aplicaciones que implementan el contrato de búsqueda y que el usuario puede seleccionar para cambiar el ámbito de su consulta.

De la misma forma, si queremos compartir información con otra aplicación utilizaremos el acceso de compartir para hacer disponible la información que estamos viendo en ese momento a otras aplicaciones que implementen el contrato de compartir y que acepten el mismo tipo de información que estamos compartiendo.

10 Contrato de búsqueda

Para que los usuarios puedan realizar búsquedas en nuestra aplicación tenemos que agregar el contrato de búsqueda. La forma más sencilla de hacerlo es utilizando la plantilla de elemento de contrato de búsqueda que nos automatiza una serie de acciones. Para añadir este elemento tenemos que ir al proyecto de la aplicación y usar Añadir > Nuevo elemento y seleccionar en el cuadro de diálogo el elemento **Contrato de búsqueda**.

Al realizar esto, Visual Studio modifica el fichero de manifiesto de la aplicación para añadir la declaración de búsqueda y además en una aplicación JavaScript, se agregan los ficheros searchResults.html, searchResults.css, y searchResults.js. Una vez hecho esto, solo tenemos que agregar la referencia al fichero JS en la página default.html y nuestra aplicación estará disponible para hacer búsquedas desde cualquier punto del sistema. En el caso de las aplicaciones XAML, el fichero que Visual Studio agrega al proyecto es SearchResults.xaml.

Las plantillas proporcionan la funcionalidad básica, pero tendremos que personalizarlas para ofrecer al usuario la mejor experiencia posible. Sin duda, lo que tendremos que hacer es modificar la vista de resultados y el código JavaScript o C# para adaptar los nombres de las propiedades al de nuestro modelo de datos. Además, tendremos que agregar también el código para navegar a la página de detalle desde los resultados, ya que por defecto viene comentada. Opcionalmente podremos añadir filtros en la página de resultados de forma que se pueda mejorar la consulta realizada.

Otra característica que podemos añadir para completar el contrato de búsqueda es ofrecer sugerencias para ayudar al usuario a encontrar algo rápidamente sin tener que ir a la vista de resultados. La forma más simple de conseguirlo es agregando el controlador del evento **SuggestionsRequested** (Código 9).

JavaScript

```
appModel.Search.SearchPane.getForCurrentView().onsuggestionsrequested = function
(eventObject) {
    var text = eventObject.queryText.toLowerCase();
    var suggestionList = ["JavaScript", "Windows 8", "XAML", "Metro", "C#"];

    suggestionList.forEach(function (term) {
        if (term.indexOf(text) == 0) {

eventObject.request.searchSuggestionCollection.appendQuerySuggestion(term);
        }
    });
};
```

C#

```
void OnSuggestionsRequested(SearchPane sender,
SearchPaneSuggestionsRequestedEventArgs args)
{
    string query = args.QueryText.ToLower();
    string[] suggestions = { "JavaScript", "Windows 8", "XAML", "Metro", "C#" };
```

```
    foreach(var term in suggestions)
    {
        if (term.StartsWith(query))
            args.Request.SearchSuggestionCollection.AppendQuerySuggestion(term);
    }
}
```

Código 9 Agregar sugerencias de búsqueda

11 Contrato de compartir

El contrato de compartir, como su nombre indica, nos permite compartir información con otras aplicaciones. En el ejemplo que estamos utilizando, al usuario podría interesarle compartir la imagen de la portada, la descripción o una dirección URL donde ver el contenido online. El proceso de compartir siempre se produce entre una aplicación origen, la que comparte el contenido, y una aplicación destino, la que lo recibe.

Para que nuestra aplicación pueda compartir contenido debemos añadir un controlador del evento **DataRequested** del objeto **DataTransferManager** que se ha asignado a la ventana actual (Código 10). Este evento se dispara cuando un usuario pulsa el acceso Compartir y en ese momento tenemos que obtener el objeto **DataPackage** para establecer los datos que queremos compartir con la aplicación de destino.

En el objeto **DataPackage** podemos establecer unas propiedades comunes como el título (*title*), descripción (*description*) y una imagen (*thumbnail*). Y mediante un conjunto de métodos podemos insertar texto, HTML, URI, Bimaps y otros elementos. Cuantos más formatos compartamos más aplicaciones habrá que consuman nuestra información.

Javascript

```
var dataTransferManager =
Windows.ApplicationModel.DataTransfer.DataTransferManager.getForCurrentView();
dataTransferManager.addEventListener("datarequested", function (e) {
   var dataPackage = e.request.data;
  dataPackage.properties.title = "DotNetMania";
  dataPackage.properties.description = "Editorial del número " + item.number;

  dataPackage.setText(item.description);
  dataPackage.setHtml(item.HTMLdescription);
  dataPackage.setUri(item.uri);});

  var image = document.querySelector(".item-image");
  var uri = image.getAttribute("src");
  uri = new Windows.Foundation.Uri(image.src); // Remote image

  var stReference =
Windows.Storage.Streams.RandomAccessStreamReference.createFromUri(uri);
  request.data.properties.thumbnail = stReference;
  var deferral = request.getDeferral();
  request.data.setBitmap(stReference);
  deferral.complete();
}
```

C#

```
void OnDataRequested(DataTransferManager sender, DataRequestedEventArgs args)
{
  var request = args.Request;
  request.Data.Properties.Title = _item.Title;
  request.Data.Properties.Description = "Editorial del número " + _item.number;
```

```
  request.Data.SetText(_item.Description.ToString());
  request.Data.SetHtml(_item.HTMLDescription.ToString());
  request.Data.SetURI(_item.Uri);

  string url = _item.GetImageUri();
  var uri = new Uri(item.url);
  var stReference = RandomAccessStreamReference.CreateFromUri(uri);
  request.Data.Properties.Thumbnail = stReference;
  var deferral = request.GetDeferral();
  request.Data.SetBitmap(stReference);
  deferral.Complete();
}
```

Código 10 El contrato compartir (origen)

12 Preferencias

Por último, vamos a ver como podemos implementar en nuestra aplicación un panel de configuración. La barra de accesos, además del acceso compartir y buscar, contiene un acceso de Configuración que permite modificar la configuración de la aplicación. Si seleccionamos este acceso, el sistema nos ofrece por defecto el comando de permisos para poder habilitar el acceso a determinadas características, como puede ser el acceso a la webcam, el micrófono o geolocalización. Nosotros podemos añadir nuevos comandos para mostrar por ejemplo la página de "Acerca de" o una página para que el usuario pueda modificar parámetros de configuración de nuestra aplicación.

En JS tenemos que agregar un controlador del evento **onsettings**. El objeto que recibimos en este evento contiene la propiedad **detail** que a su vez contiene la propiedad **applicationcommands**. Esta propiedad la podemos utilizar para establecer mediante un array los comandos que queremos que aparezcan en el panel de configuración, indicando la página que queremos que se abra y el titulo. Una vez hemos establecido esta propiedad tenemos que realizar una llamada a **populateSettings** (Código 11).

En XAML tenemos que hacer algo similar suscribiéndonos al evento **CommandsRequested**, pero en lugar utilizar un array, tenemos que añadir a la colección **ApplicationCommands** un objeto **SettingsCommand** para cada uno de los comandos, en los que establecemos el id, el titulo y el controlador para el evento que se lanza cuando el usuario selecciona el comando.

JavaScript

```
WinJS.Application.onsettings = function (e) {
    e.detail.applicationcommands = {
        "about": { href: "About.html", title: "About"},
    };
    WinJS.UI.SettingsFlyout.populateSettings(e);
}

<div id="defaultsDiv" data-win-control="WinJS.UI.SettingsFlyout" aria-label="App
defaults settings flyout"
        data-win-options="{width:'wide'}">
    <div class="win-header" style="background-color:#dbf9ff">
        <button type="button" onclick="WinJS.UI.SettingsFlyout.show()" class="win-
backbutton"></button>
        <div class="win-label">Defaults</div>
        <img src="images/smallTile.png" style="position: absolute; right: 40px;"/>
    </div>
    <div class="win-content">
        Aplicación Ejemplo DotNetMania
```

```
    </div>
</div>
```

C#

```
void OnCommandsRequested(SettingsPane sender,
SettingsPaneCommandsRequestedEventArgs args)
{
    var about = new SettingsCommand("about", "About", (handler) =>
        {
            var settings = new SettingsFlyout();
            settings.ShowFlyout(new AboutUserControl());
        });

    args.Request.ApplicationCommands.Add(about);
}
```

Código 11 Controlador de evento para añadir comandos personalizados al panel de configuración

13 Resumen

Hasta aquí este recorrido por los conceptos esenciales para iniciarnos en el desarrollo de aplicaciones Windows 8 UI Style. Hemos visto la forma más sencilla de crear una aplicación basándonos en la plantilla cuadrícula de Visual Studio, consultando los datos desde un servicio web y mediante la implementación de contratos hemos integrado la aplicación con el sistema, permitiendo que se pueda buscar dentro de nuestra aplicación en cualquier momento y desde cualquier lugar, así como compartir información con otras aplicaciones.

Hemos visto como el nuevo modelo de gestión se centra en la aplicación que está en primer plano para asegurar una perfecta experiencia de usuario, así como el diseño de las mismas se centra en el contenido relegando las acciones a un segundo plano. Y hemos visto como con Windows 8 es el propio sistema quien gestiona el tiempo de vida de las aplicaciones y como debemos preparar nuestras aplicaciones para ofrecer una experiencia de a los usuarios.