



Diapositiva 1




Día 6:
Crear aplicaciones Windows




1

Diapositiva 2



Objetivos para el día

- Al final de este día, usted podrá:
 - Crear una aplicación Windows y utilizar controles generales
 - Entender la funcionalidad de los archivos de configuración



2

Diapositiva 3

Contenidos del día

- 1. Crear Forms
- 2. Usar cuadros de diálogo comunes
- 3. Usar cuadros de diálogo personalizados
- 4. Usar archivos de configuración

3

Diapositiva 4

1. Crear Forms

Preguntas de la sección

- Preguntas resueltas
 - ¿Cómo crear una Windows Form?
 - ¿Cómo utilizar los controles sobre las formas?

4

Diapositiva 5

Aplicaciones Windows

- Una ventana es un componente clave de Windows
 - Se configura a través de atributos
 - SendMessage es la función más importante
 - No está realmente orientada a objetos
- Una clase base común: Control
- La clase Form se deriva de Control
- Nuevos Controls y Forms a través de herencia



5

Windows Forms es la nueva plataforma para el desarrollo de aplicaciones Microsoft Windows, basada en .NET Framework. Este marco proporciona un conjunto de clases claro, orientado a los objetos y ampliable que le permite desarrollar aplicaciones Windows enriquecidas. Además, Windows Forms pueden actuar como la interfaz local en una solución distribuida de múltiples capas.

Una ventana es el componente central de Windows

Una ventana en una Aplicación Windows se representa mediante un objeto Form. A nivel de sistema operativo, las ventanas se comunican utilizando mensajes. Estos mensajes se envían usando el método SendMessage. Este método no es parte de una Aplicación Windows debido a que reside en un componente no administrado que es parte de los componentes del sistema operativo Windows y usted no necesita preocuparse sobre esta comunicación, excepto que necesite enviar mensajes personalizados entre ventanas. A nivel de aplicación, usted trabaja únicamente en un ambiente orientado a objetos, instancia objetos Form y trabaja con sus métodos y propiedades, además de que agrega la lógica a su aplicación.

Controles y formas nuevos a través de la herencia

Un objeto Form y los controles que contiene se heredan de sus clases correspondientes. un formulario se hereda de System.Windows.Forms.Form y los controles de System.Windows.Forms.Control, aunque cada control tiene su propia clase que se deriva de System.Windows.Forms.Control, por ejemplo un control TextBox se deriva de la clase System.Windows.Forms.TextBox.

Ejemplo. Crear un objeto de formulario.

```
...
public class Form2 : System.Windows.Forms.Form
{
    public Form2()
    {
        // Constructor. The controls are added here.
    }
    // Body of the form, events, methods, etc.
}
```

Ejemplo. Crear un control sobre el formulario.

Este formulario contiene dos controles: un control TextBox y un control Button.

```
...
public class Form2 : System.Windows.Forms.Form
{
    private System.Windows.Forms.TextBox textBox1;
    private System.Windows.Forms.Button button1;
    public Form2()
    {
        // Constructor. The controls are added here.
        this.textBox1 = new System.Windows.Forms.TextBox();
        this.button1 = new System.Windows.Forms.Button();

        // Temporarily suspends the layout logic for the form.
        this.SuspendLayout();
        //
        // textBox1
    }
}
```

```

//
this.textBox1.Location = new System.Drawing.Point(16, 16);
this.textBox1.Name = "textBox1";
this.textBox1.TabIndex = 0;
this.textBox1.Text = "textBox1";
//
// button1
//
this.button1.Location = new System.Drawing.Point(136, 16);
this.button1.Name = "button1";
this.button1.TabIndex = 1;
this.button1.Text = "button1";
//
// Form2
//
this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
this.ClientSize = new System.Drawing.Size(292, 273);

// Add controls to the form
this.Controls.AddRange(new System.Windows.Forms.Control[] {
this.button1, this.textBox1});
this.Name = "Form2";
this.Text = "Form2";
// Resumes normal layout logic of the form.
this.ResumeLayout(false);
}
// Body of the form, events, methods, etc.
}

```

Diapositiva 6

Aplicaciones Windows

- Objeto de la aplicación
 - Métodos y propiedades estáticos para administrar una aplicación
- Punto de inicio de una Aplicación Windows

```
[STAThread]
static void Main()
{
    System.Windows.Forms.Application.Run(new Form1());
}
```

6

Objeto Application

La clase Application tiene métodos para iniciar y detener aplicaciones y subprogramas, y para procesar mensajes Windows. Invocar Run para iniciar el circuito de mensaje de la aplicación en el subproceso actual y, opcionalmente, hacer un formulario visible. Invocar Exit o ExitThread para detener el circuito de un mensaje. Invocar DoEvents para procesar mensajes mientras que su programa está en circuito.

Ejemplo

El método Main invoca Run para iniciar la aplicación, la cual crea la forma, listBox1 y button1. Cuando el usuario hace clic en button1, el método button1_Click agrega los números uno a tres al cuadro de lista y muestra un MessageBox. Si el usuario hace clic en No en el MessageBox, el método button1_Click agrega otro número a la lista. Si el usuario hace clic en Sí, la aplicación invoca Exit para procesar todos los mensajes restantes en la cola de espera y después para salir.

El ejemplo supone que listBox1 y button1 han sido instanciados y colocados en un formulario .

```
public static void Main(string[] args) {
    // Starts the application.
    System.Windows.Forms.Application.Run(new Form1());
}
```

```
protected void button1_Click(object sender, System.EventArgs e)
{
    // Populates a list box with three numbers.
    int i = 3;
    for(int j=1; j<=i; j++)
    {
        listBox1.Items.Add(j);
    }

    // Checks to see whether the user wants to exit the application.
    // If not, adds another number to the list box.
    while (MessageBox.Show("Exit application?", "", MessageBoxButtons.YesNo) == DialogResult.No)
    {
        // Increment the counter and add the number to the list box.
        i++;
        listBox1.Items.Add(i);
    }

    // The user wants to exit the application. Close everything down.
    Application.Exit();
}
```

Diapositiva 7

Crear Forms

- Realmente orientada a objetos
- Muy familiares para los programadores de Visual Basic
 - Form
 - Crear instancia
 - Invocar el método Show o ShowDialog
 - Utilizar propiedades y métodos
 - Control
 - Crear instancia
 - Agregar a los contenedores la colección de Controles
 - Utilizar propiedades y métodos

7

¿Qué es un Formulario?

un formulario es un espacio de la propiedad en la pantalla, generalmente rectangular, que puede usar para presentar información al usuario o aceptar información de entrada del usuario. Las formas pueden ser ventanas estándar, ventanas de interfaz múltiple de documento (MDI), cuadros de diálogo o superficies de presentación para rutinas gráficas. Además, los formularios son controles, debido a que se heredan desde la clase Control.

Crear una instancia

Cada formulario en su aplicación es una clase y usted sólo necesita instanciar la clase form requerida y utilizarla.

```
FCient MyForm = new FCient();
```

```
MyForm.ClientId = 23; // Public property.
```

```
MyForm.Text = "Hello"; // Change the title.
```

```
MyForm.Refresh(); // Redraw itself.
```

Mostrar un formulario

Los objetos Form contienen dos métodos para mostrarse: Show y ShowDialog. Show: muestra un formulario al usuario. Mostrar el control es equivalente a configurar la propiedad Visible en true. Después de invocar el método Show, la propiedad Visible devuelve un valor true hasta que se invoca el método Hide.

```
FCient MyForm = new FCient();
```



```
MyForm.ClientId = 23; // Public property.
MyForm.Text = "Hello"; // Change the title.
MyForm.Show();
```

ShowDialog: muestra un formulario como un cuadro de diálogo modal. Cuando se muestra un formulario de manera modal, no puede ocurrir ningún ingreso de información (ya sea utilizando el teclado o haciendo clic con el mouse) con la excepción de los objetos del formulario modal. El programa debe ocultar o cerrar el formulario modal (con frecuencia en respuesta a alguna acción del usuario) antes de que pueda ocurrir el ingreso a algún formulario. Los formularios que aparecen de manera modal generalmente se utilizan como cuadros de diálogo en una aplicación.

```
public void ShowMyDialogBox()
{
    FClient MyForm = new FClient();
    MyForm.ClientId = 23; // Public property.
    MyForm.Text = "Hello"; // Change the title.
    MyForm.Show();
    // Show FClient as a modal dialog and determine if DialogResult = OK.
    if (MyForm.ShowDialog(this) == DialogResult.OK)
        // OK. Do something
    else
        // Cancel. Do something
    MyForm.Dispose();
}
```

Propiedad DialogResult

El resultado de un diálogo de un formulario es el valor que se devuelve desde el mismo cuando aparece como un diálogo modal. Si aparece como un cuadro de diálogo, al configurar esta propiedad con un valor desde la enumeración System.Windows.Forms.DialogResult (Abort, Cancel, Ignore, No, None, Ok, Retry, Yes) establece el valor del resultado del diálogo para el formulario, oculta el diálogo modal y devuelve el control a la forma que se invoca. Esta propiedad por lo general se configura a través de la propiedad DialogResult del control Button en la forma. Cuando el usuario hace clic en el control Button, el valor asignado a la propiedad DialogResult de Button se asigna a la propiedad DialogResult de la forma.

Cuando aparece un formulario como un cuadro de diálogo modal, hacer clic en el botón Close (el botón con una X en la esquina superior derecha de la forma) hace que la forma se oculte y la propiedad DialogResult se configure a DialogResult.Cancel.

Diapositiva 8

Crear Forms

- Se heredan de Control o Form
- La forma del control de enviar un mensaje: un evento
- Delegados
 - Apuntadores de función orientados a objetos
 - Los delegados están protegidos contra el tipo
- Los eventos se basan en delegados
 - Actúan como puntos de conexión
 - Es posible tener más de un manejador



8

Clase control

Define la clase base para los controles, los cuales son componentes con una representación visual. La clase Control implementa una funcionalidad muy básica requerida por las clases que presentan información al usuario. Maneja el ingreso de información del usuario a través del teclado y de dispositivos de señalamiento. Maneja el enrutamiento y la seguridad de los mensajes. Define los límites de un control (su posición y tamaño), aunque no implementa su dibujo. Proporciona un manejador (handle) de ventana (hWnd). Los controles Windows Forms utilizan propiedades de ambiente de manera que los controles hijos puedan parecer similares al ambiente que los rodea. Una propiedad de ambiente es una propiedad de control que, si no se configura, se recupera del control padre.

Declaración de evento dentro de una Form

El siguiente ejemplo muestra la manera en que se agrega un manejador de evento a un control Button en un Formulario.

Cuando se define el control en el constructor de la clase Form, debe asignar el manejador.

...

```
this.button1 = new System.Windows.Forms.Button();  
this.button1.Location = new System.Drawing.Point(136, 16);  
this.button1.Name = "button1";
```

```
this.button1.TabIndex = 1;  
this.button1.Text = "button1";  
this.button1.Click += new System.EventHandler(this.button1_Click);  
...
```

La clase del Formulario debe contar con el método button1_Click.

```
private void button1_Click(object sender, System.EventArgs e)  
{  
    // Add code to handle the event.  
}
```

El manejador de evento recibe un argumento del tipo EventArgs que contiene los datos relacionados con este evento.

Diapositiva 9

Crear Forms – Controles contenidos

- Agregar un control a un formulario
 - Crear una instancia de un control
 - Establecer atributos de un control (Text, BackColor,...)
 - Establecer la ubicación y las dimensiones del control
 - Agregar manejadores de evento al control
 - Agregar el control a la colección de controles de la forma
- Botones, Controles de texto, Controles de lista, Menús, Cuadros de diálogo comunes y más

9

El Diseñador de Windows Forms de Visual Studio .Net proporciona una solución de desarrollo rápida para crear aplicaciones Windows. El primer paso en el diseño de un formulario es establecer sus propiedades. Los menús, cuadro de diálogo, barras de estado y barras de herramientas son herramientas que le permiten exponer la funcionalidad a sus usuarios o alertarlos para información importante en su aplicación.

La mayoría de las formas se diseñan al agregar controles a la superficie de la forma para definir la interfaz. Un control es un componente en un formulario utilizado para mostrar la información o aceptar el ingreso de información del usuario.

Para arrastrar un control a un formulario


1. Abra el formulario.
2. En la caja de herramientas haga clic en el control que desea y arrástrelo sobre el formulario. El control se agrega en la ubicación especificada en su tamaño predeterminado. Puede hacer doble clic en un control en la caja de herramientas para agregarlo en la esquina superior izquierda del formulario en su tamaño predeterminado.

Agregar controles de manera dinámica



Para agregar controles de manera dinámica a un formulario durante el tiempo de ejecución. En el siguiente ejemplo, se agregará un control TextBox forma cuando se haga clic en el control Button. El siguiente procedimiento supone la existencia de un formulario con un control Button, Button1, ya definido.

```
private void button1_Click(object sender, System.EventArgs e)
{
    TextBox myText = new TextBox();
    myText.Location = new Point(25,25);
    myText.Text = "Hello World!!!"
    this.Controls.Add (myText); // Add the control to the Controls collection of the
    form.
}
```

Diapositiva 10

 Crear Forms – Controles contenidos

- Controles generales por función
 - Edición de texto
 - TextBox, RichTextBox
 - Texto mostrado (sólo lectura)
 - Label, LinkLabel, StatusBar
 - Selección de una lista
 - CheckedListBox, ComboBox, Listbox, ListView, TreeView, NumericUpDown
 - Controles del menú
 - MainMenu, ContextMenu
 - Comandos
 - Button, LinkLabel, ToolBar
 - Establecimiento de valores
 - CheckBox, CheckedListBox, RadioButton,


10

Hay más controles que se pueden incluir en un formulario .

Edición de texto

TextBox: Muestra texto ingresado durante el diseño que pueden editar los usuarios durante el tiempo de ejecución, o lo pueden cambiar de manera programática.

RichTextBox : Permite que el texto a parezca con el formateo en texto plano o formato de texto enriquecido (RTF).

Presentación de texto (sólo lectura)

Label : Muestra texto que los usuarios no pueden editar directamente.

LinkLabel : Muestra texto como un vínculo tipo Web y dispara un evento cuando el usuario hace clic en un texto especial. Por lo general este texto es un vínculo a otra ventana o a un sitio Web.

StatusBar : Muestra la información acerca del estado actual de la aplicación utilizando una ventana enmarcada, por lo general aparece en la parte inferior de la forma padre.

Ejemplo de uso de StatusBar

Con frecuencia, un programa le pedirá que actualice los contenidos de los paneles de la barra de estado dinámicamente durante el tiempo de ejecución, con base en los cambios al estado de la aplicación u otra interacción del usuario. Esta es un formulario común para indicar a los usuarios que teclas tales como CAPS LOCK, NUM LOCK, o SCROLL LOCK están habilitadas, o

para proporcionar la fecha o un reloj como una referencia conveniente. El área programable dentro del control StatusBar consiste en instancias de la clase StatusBarPanel. Éstas se agregan durante el periodo del diseño a través del Editor de colección de StatusBarPanel, y durante el tiempo de ejecución a través de la clase StatusBarPanelCollection.

Este método agrega un panel a StatudBar.

```
public void CreateStatusBarPanels()
{
    // Create panels and set text property.
    statusBar1.Panels.Add("One");
    statusBar1.Panels.Add("Two");
    statusBar1.Panels.Add("Three");
}

This method update the first panel of the status bar with the current date.
private void UpdateStatusBar()
{
    statusBar1.Panels[0].Text = DateTime.Now.ToShortTimeString();
    statusBar1.Panels[1].Text = "Hello";
    statusBar1.Panels[2].Text = "World";
}
```

Selección desde una lista

CheckedListBox : Muestra una lista desplazable de elementos, cada una acompañada por un cuadro de verificación.

ComboBox : Muestra una lista desplegable de elementos.

DomainUpDown : Muestra una lista de elementos de textos a través de la cual se pueden desplazar los usuarios con los botones hacia arriba y hacia abajo.

ListBox : Muestra una lista de elementos de texto y gráficos (iconos).

ListView : Muestra los elementos en una de cuatro formas diferentes. Las vistas incluyen sólo texto, texto con iconos pequeños, texto con iconos grandes y una vista de reporte. (Igual que Windows Explorer)

NumericUpDown : Muestra una lista de numerables a través de la cual se pueden desplazar los usuarios con los botones hacia arriba y hacia abajo.

TreeView : Muestra una colección jerárquica de objetos de nodos que puede consistir en texto con cuadros de verificación o iconos opcionales. (Igual que Windows Explorer)

Controles de menús

MainMenu : Proporciona una interfaz durante el tiempo de diseño para crear menús.

ContextMenu : Implementa un menú que aparece cuando el usuario hace clic con el botón alterno en un objeto.

Ejemplo

Este método crea un menú principal y lo agrega al objeto form.

```
public void CreateMyMainMenu()
{
    // Create an empty MainMenu derived of class System.Windows.Forms.MainMenu.
    MainMenu mainMenu1 = new MainMenu();
    MenuItem menuItem1 = new MenuItem();
    MenuItem menuItem2 = new MenuItem();
}
```

```

menuItem1.Text = "File";
menuItem2.Text = "Edit";
// Add two MenuItem objects to the MainMenu.
mainMenu1.MenuItems.Add(menuItem1);
mainMenu1.MenuItems.Add(menuItem2);

// Bind the MainMenu to Form1.
// The form object have a property to get or set the MainMenu that is displayed in the form.
Menu = mainMenu1;
}
This method creates a context menu and add it to the form object.
public void AddContextMenuAndItems()
{
    ContextMenu mnuContextMenu = new ContextMenu();

    // Add ContextMenu to the Form
    // The form object have a property to get or set the ContextMenu that is displayed in the form.
    this.ContextMenu = mnuContextMenu;
    MenuItem mnulItemNew = new MenuItem();
    MenuItem mnulItemOpen = new MenuItem();

    mnulItemNew.Text = "&New";
    mnulItemOpen.Text = "&Open";
    mnuContextMenu.MenuItems.Add(mnulItemNew);
    mnuContextMenu.MenuItems.Add(mnulItemOpen);
    MenuItem mnulItemOpenWith = new MenuItem();
    mnulItemOpenWith.Text = "Open &With...";
    mnulItemOpen.MenuItems.Add(mnulItemOpenWith);
}

```

Comandos

Button: Se utiliza para iniciar, detener o interrumpir un proceso.

LinkLabel : Muestra texto con un vínculo estilo Web y dispara un evento cuando el usuario hace clic en un texto especial. Por lo general el texto es un vínculo a otra ventana o sitio Web.

NotifyIcon : Muestra un icono en el área de notificación del estado de la barra de tareas que representa una aplicación que se ejecuta en el fondo.

ToolBar : Contiene una colección de controles de botón.

Configuración de valores

CheckBox : Muestra un cuadro de verificación y una etiqueta para texto. Por lo general se usa para configurar opciones.

CheckedListBox : Muestra una lista desplazable de elementos, cada una acompañada de un cuadro de verificación.

RadioButton : Muestra un botón que se puede activar o desactivar.

Trackbar : Permite que los usuarios establezcan valores en una escala al mover un "pulgar" en la escala.

Diapositiva 11



2. Usar cuadros de diálogo comunes

Preguntas de la sección

■ Preguntas resueltas

- ¿Qué es un cuadro de diálogo común?
- ¿Cómo utilizamos un cuadro de diálogo común?



Diapositiva 12

Cuadros de diálogo comunes

■ CommonDialog Class

■ OpenFileDialog, SaveFileDialog

- Derivados de FileDialog
- Ofrece el método OpenFileDialog

■ FontDialog

- Presenta un evento Apply para la funcionalidad de vista previa

■ ColorDialog

■ PageSetupDialog

■ PrintDialog

12

ColorDialog

Representa un cuadro de diálogo común que muestra los colores disponibles junto con los controles que permiten al usuario definir los colores personalizados.

PageSetpuDialog

Representa un cuadro de diálogo que permite a los usuarios manipular las configuraciones de la página, incluyendo márgenes y la orientación del papel.

PrintDialog

Permite que los usuarios seleccionen una impresora y elijan qué porciones del documento imprimir.

Example of OpenFileDialog:

```
public void OpenFile()
{
    System.IO.Stream myStream;
    OpenFileDialog openFileDialog1 = new OpenFileDialog();

    // Set the initial directory
    openFileDialog1.InitialDirectory = "c:\\\" ;
    // Only display text files
```

```
openFileDialog1.Filter = "txt files (*.txt)|*.txt|All files (*.*)|*.*" ;
openFileDialog1.FilterIndex = 2 ;
openFileDialog1.RestoreDirectory = true ;
if(openFileDialog1.ShowDialog() == DialogResult.OK)
{
    // Verify if the user select any file.
    if((myStream = openFileDialog1.OpenFile())!= null)
        MessageBox.Show( myStream.ToString() ); // Display a file name selected.
}
}
```

Diapositiva 13

2. Usar cuadros de diálogo personalizados

Preguntas de la sección

- Preguntas resueltas
 - ¿Qué es un cuadro de diálogo personalizado?
 - ¿Cómo utilizamos un cuadro de diálogo personalizado?



Diapositiva 14

Cuadros de diálogo personalizados

- Derivados de la clase Form
- Utilice ShowDialog para abrir un cuadro de diálogo
 - Devuelve el valor del DialogResult
 - Cómo se cierra el cuadro de diálogo
 - Aceptar, Cancelar, Sí, ...
- Propiedades AcceptButton, CancelButton
 - Manejo adicional del botón Aceptar y Cancelar
- Devolver los valores de ingreso a través de
 - Controles públicos
 - Propiedades públicas
- Tipo de cuadro de diálogo especial: MessageBox

14

¿Qué es un cuadro de diálogo personalizado ?

Un cuadro de diálogo es un formulario derivada de System.Windows.Forms que puede crear durante el tiempo de ejecución para obtener una respuesta del usuario. La forma tiene una propiedad llamada DialogResult que obtiene o establece el resultado del cuadro de diálogo para la forma.

Propiedad DialogResult

Si el valor de esta propiedad se configura para cualquier cosa que no sea None, y si la forma padre aparece a través del método ShowDialog, hacer clic en el botón cierra la forma padre sin tener que colgarse de ningún evento. La propiedad DialogResult de la forma después se configura para DialogResult del botón cuando se hace clic en el botón.

Por ejemplo, para crear un cuadro de diálogo "Yes/No/Cancel", simplemente agregue tres botones y configure sus propiedades DialogResult como Yes, No y Cancel.

Ejemplo

```
public void CreateMyForm()
{
    // Create a new instance of the form.
    Form form1 = new Form();
    // Create two buttons to use as the accept and cancel buttons.
    Button button1 = new Button ();
```

```

Button button2 = new Button ();

// Set the text of button1 to "OK".
button1.Text = "OK";
// Set the position of the button on the form.
button1.Location = new Point (10, 10);
// Set the text of button2 to "Cancel".
button2.Text = "Cancel";
// Set the position of the button based on the location of button1.
button2.Location = new Point (button1.Left, button1.Height + button1.Top + 10);
// Make button1's dialog result OK.
button1.DialogResult = DialogResult.OK;
// Make button2's dialog result Cancel.
button2.DialogResult = DialogResult.Cancel;
// Set the caption bar text of the form.
form1.Text = "My Dialog Box";

// Define the border style of the form to a dialog box.
form1.FormBorderStyle = FormBorderStyle.FixedDialog;
// Set the accept button of the form to button1.
form1.AcceptButton = button1;
// Set the cancel button of the form to button2.
form1.CancelButton = button2;
// Set the start position of the form to the center of the screen.
form1.StartPosition = FormStartPosition.CenterScreen;
// Add button1 to the form.
form1.Controls.Add(button1);
// Add button2 to the form.
form1.Controls.Add(button2);
// Display the form as a modal dialog box.
form1.ShowDialog();

// Determine if the OK button was clicked on the dialog box.
if (form1.DialogResult == DialogResult.OK)
{
    // Display a message box indicating that the OK button was clicked.
    MessageBox.Show("The OK button on the form was clicked.");
    // Optional: Call the Dispose method when you are finished with the dialog box.
    form1.Dispose();
}
else
{
    // Display a message box indicating that the Cancel button was clicked.
    MessageBox.Show("The Cancel button on the form was clicked.");
    // Optional: Call the Dispose method when you are finished with the dialog box.
    form1.Dispose();
}
}

```

Propiedades AcceptButton, CancelButton

La propiedad AcceptButton le permite designar una acción predeterminada para que ocurra cuando el usuario presiona la tecla INTRO en su aplicación. El botón cancel para un formulario es el control de botón en el que se hace clic cuando el usuario presiona la tecla ESC.

Clase MessageBox

La Clase MessageBox es un cuadro de mensaje que puede contener texto, botones y símbolos que informan e instruyen al usuario. También es posible obtener una respuesta del usuario al configurar el parámetro buttons con los valores AbortRetryIgnore, OKCancel, RetryCancel, YesNo y YesNoCancel.

Ejemplo:

```
public void MsgBox()
{
    DialogResult result;
    // Displays the MessageBox.
    result = MessageBox.Show("Message?", "Caption", MessageBoxButtons.YesNo,
        MessageBoxIcon.Question, MessageBoxDefaultButton.Button1);
    if(result == DialogResult.Yes)
    {
        // User press button Yes.
    }
}
```

Diapositiva 15



4. Usar archivos de configuración

Preguntas de la sección

- Preguntas resueltas
 - ¿Qué es un archivo de configuración?
 - ¿Cómo usar los archivos de configuración?



Diapositiva 16

Archivos de configuración

- <aplicación>.config
 - Utilizado para una configuración específica de la aplicación
 - Debe estar en la misma ubicación que el .EXE principal de la aplicación
- XML-Files
 - Legibles y editables por el humano

```
<configuration>
  <appSettings>
    <add key="Application Name" value="MyApplication" />
  </appSettings>
</configuration>
```

```
public void ReadMyAppSettings() {
    string appName = ConfigurationSettings.AppSettings["Application Name"];

    Console.WriteLine();
    Console.WriteLine("Reading AppSettings");
    Console.WriteLine("Application Name: " + appName);
}
```

16

Se encuentra un archivo de configuración para una aplicación alojada por un host ejecutable en el mismo directorio que la aplicación. El nombre del archivo de configuración es el nombre de la aplicación con una extensión .config. Por ejemplo, una aplicación llamada myApp.exe se puede asociar con un archivo de configuración llamado myApp.exe.config.

Clase ConfigurationSettings

Esta clase proporciona acceso a los elementos de configuración en una sección específica de configuración. La propiedad AppSettings obtiene los elementos de configuración en la sección de configuración del Elemento <appSettings>.

Sección AppSettings

.NET Framework proporciona una sección de configuración predefinida llamada **appSettings**. El siguiente ejemplo muestra la declaración de la sección **appSettings** como aparecen en el archivo de configuración de la máquina (Machine.config).

```
<configuration>
```

```

    <!-- Configuration section declarations. -->
    <configSections>
        <section name="appSettings"
type="System.Web.Configuration.NameValueSectionHandler"/>
    </configSections>
</configuration>

```

Ejemplo.

En este ejemplo el nombre del ejecutable de la aplicación es MyApplication.exe.

Archivo MyApplication.exe.config.

```

<configuration>
    <appSettings>
        <add key="Application Name" value="MyApplication" />
    </appSettings>
</configuration>

```

Este método da como resultado el valor del parámetro "Application Name".

```

public void ReadMyAppSettings()
{
    string appName = ConfigurationSettings.AppSettings["Application Name"];
    Console.WriteLine();
    Console.WriteLine("Reading AppSettings");
    Console.WriteLine("Application Name: " + appName);
}

```

Crear una sección de configuración personalizada

También puede definir su propia sección que utilice el mismo manejador de configuración que la sección <appSettings>. Por ejemplo:

```

<configuration>
    <configSections>
        <sectionGroup name="myGroup">
            <sectionGroup name="nestedGroup">
                <section name="mySection"
type="System.Configuration.NameValueSectionHandler" />
            </sectionGroup>
        </sectionGroup>
    </configSections>

```

```
</configSections>
<myGroup>
  <nestedGroup>
    <mySection>
      <add key="key_one" value="1" />
      <add key="key_two" value="2" />
    </mySection>
  </nestedGroup>
</myGroup>
</configuration>
```

Pude leer el valor de la nueva sección de configuración definida en el ejemplo anterior como sigue:

```
NameValueCollection config = (NameValueCollection)
// The method GetConfig returns configuration settings for a user-defined
configuration section.
ConfigurationSettings.GetConfig("mygroup/nestedgroup/mysection");
Response.Write( "The value of key_one is " + config["key_one"] );
Response.Write( "The value of key_two is " + config["key_two"] );
```

Diapositiva 17

Resumen de recursos

- Windows Forms <http://msdn.microsoft.com/library/en-us/vbcon/html/vbconintroductiontowfcforms.asp?frame=true>
- Cuadros de diálogo personalizados
<http://msdn.microsoft.com/library/en-us/cpref/html/frrfSystemWindowsFormsFormClassDialogResultTopic.asp?frame=true>
- Archivos de configuración
<http://msdn.microsoft.com/library/en-us/cpguide/html/cpconconfigurationsectionhandlers.asp?frame=true>

