

## Situación profesional 3: Incendios

Un funcionario público le ha solicitado a usted que, como pasante del área de sistemas de la dependencia estatal que él gestiona, desarrolle una aplicación para obtener resultados de los incendios registrados en el último año en las provincias detallado por departamento.

Las tareas a realizar son las siguientes:

1. Agregar a un control de tipo árbol un nodo raíz con la leyenda “Incendios”. Del nodo raíz dependen nodos con los nombres de las provincias. Y de cada nodo con el nombre de una provincia dependen nodos con los nombres de los departamentos que pertenecen a cada provincia.

2. Al seleccionar un nodo con el nombre de una provincia o el nombre de un departamento, se muestra en un control de tipo vista de detalle con dos columnas, el tipo de incendio y la cantidad de incendios registrada en la provincia o en el departamento (la cantidad de incendios obtenidos por cada lista depende del nodo seleccionado). Al seleccionar el nodo raíz se deberá mostrar los valores acumulados de todas las provincias.

3. También se muestra en un control barra de estado la cantidad total de incendios, es la suma de todos los incendios que aparecen en el control vista de detalle.

4. Cuando se selecciona el nodo raíz del control tipo árbol se inicializa el control vista de detalle y el control barra de estado.

La base de datos se llama ‘Incendios.mdb’ y contiene cuatro tablas.

La tabla ‘Provincias’ contiene las provincias participantes en la consulta, sus datos son: un número para identificar la provincia y el nombre de la provincia. La clave principal de la tabla es el número de provincia.

La tabla ‘Departamentos’ contiene los departamentos de cada provincia, sus datos son: un número para identificar al departamento, el nombre del departamento y el número de provincia al que pertenece el departamento. La clave principal de la tabla es el número de departamento.

La tabla ‘TipoIncendio’ contiene los diferentes tipos de incendio según el origen del mismo, sus datos son: un número para identificar el tipo y la descripción de cada tipo. La clave principal de la tabla es el número de lista.

La tabla ‘Incendios’ contiene la cantidad de incendios registrados por cada tipo en los respectivos departamentos, sus datos son: el número de departamento, el tipo de incendio y la

cantidad de incendios. La clave principal de la tabla está compuesta por las dos primeras columnas.

Incendios		Cantidades	
INCENDIOS		Tipo de Incendio	Cantidad
Cordoba		Natural	230
Capital		Accidental	25
Colon		Intencional	89
Rio Cuarto		Desconocido	47
Punilla			
Santa Fe			
La Pampa			

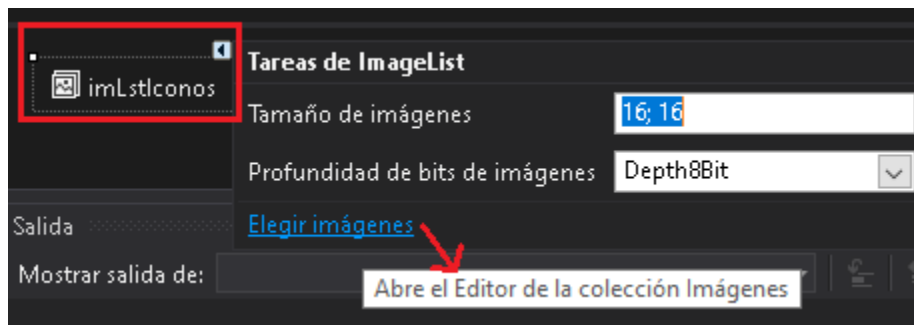
Cantidad Total de Incendios Departamento Colon: 391

## SP3/H1: Control ImageList

El control ImageList es un contenedor de imágenes. Nos permitirá mostrar los distintos tipos de carpetas cuando desarrollemos la aplicación que se nos solicitó.

Un control ImageList contiene una colección de objetos Image, a los que se puede hacer referencia mediante su índice o clave.

El control ImageList no está concebido para usarlo en solitario, sino como punto de almacenamiento central para proporcionar cómodamente imágenes a otros controles. Es un control que no se visualiza sobre el formulario, es un complemento para ser usado en combinación con otros controles que puedan visualizar iconos o imágenes, como, por ejemplo, un control PictureBox, TreeView, ListView, DataGridView, CheckBox, RadioButton, Label y algunos más. En general cualquier control que posea la propiedad "ImageList" podrá asociarse con el nombre de un control ImageList.



### Para Agregar/Quitar Imágenes

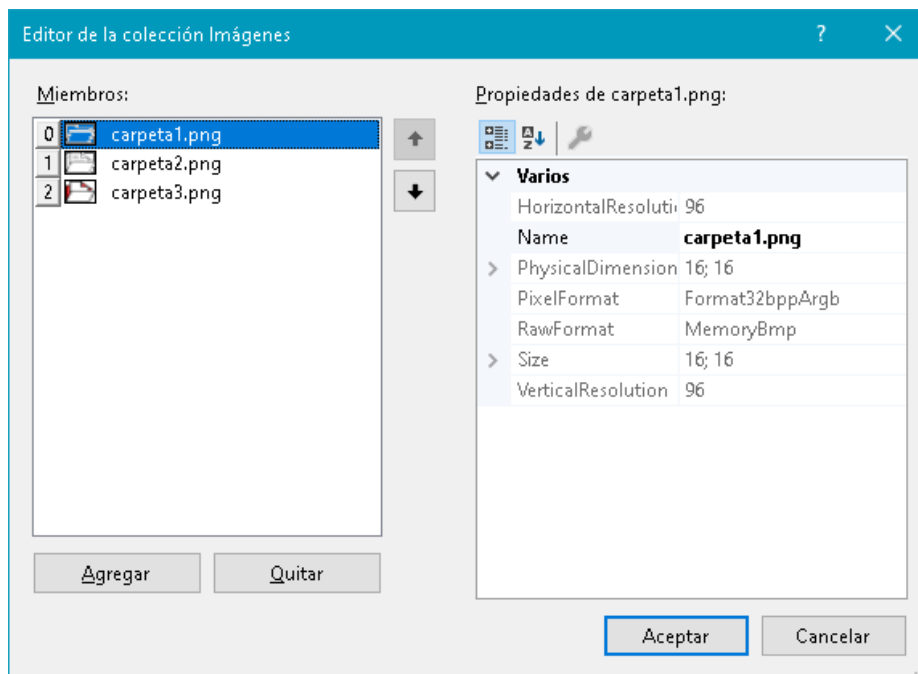
Puede agregar/quitar imágenes a un componente ImageList de varias maneras. Puede utilizar la etiqueta inteligente asociada al componente ImageList, o quizá le resulte más cómodo agregar/quitar imágenes con la ventana de Propiedades. También puede agregar imágenes utilizando un código.

#### a) Utilizando la ventana de propiedades:

Seleccione el componente ImageList o agregue uno al formulario.

En la ventana Propiedades, haga clic en el botón de puntos suspensivos (...) situado junto a la propiedad Images.

En el Editor de la colección de imágenes, haga clic en Agregar o Quitar para agregar o quitar las imágenes de la lista.



### b) Mediante la etiqueta inteligente:

Seleccione el componente ImageList o agregue uno al formulario.

Haga clic en el glifo de la etiqueta inteligente.

En el cuadro de diálogo seleccione "Elegir imágenes".

En el Editor de la colección de imágenes, haga clic en Agregar o Quitar para agregar o quitar las imágenes de la lista.

### c) Mediante Programación:

Utilice el método Add de la propiedad Images de la lista de imágenes.

El siguiente ejemplo de código requiere que tenga un formulario con un control ImageList ya agregado y también se requiere tener agregado:

```
using System.Drawing;
```

```
Image imagen = Image.FromFile("C:\\Carpeta1.png");  
imLstIconos.Images.Add(imagen);
```

“imLstIconos” es el nombre del control ImageList.

“Image” es la clase que maneja los objetos de tipo Image.

### Para Quitar Imágenes Mediante Programación

Utilice el método `RemoveAt` para quitar una sola imagen, indicando el índice de la misma, o bien, el método `Clear`, para borrar todas las imágenes de la lista de imágenes:

```
imLstIconos.Images.RemoveAt(0);  
imLstIconos.Images.Clear();
```

### **Asociar una Lista de Imágenes con un Control de Windows**

Para los controles comunes de Windows, se puede especificar un `ImageList` durante el diseño del programa, mediante el cuadro de diálogo `Propiedades`.

En tiempo de ejecución se puede especificar también un control `ImageList` utilizando la propiedad `ImageList`, como en el siguiente ejemplo:

```
chkImagen.ImageList = ImageList1;  
chkImagen.ImageIndex = 1;
```

Al usar el control `ImageList` con controles comunes de Windows, será necesario insertar todas las imágenes que se necesitarán (en orden adecuado), en el control `ImageList` antes de enlazarlo al control secundario. Una vez que `ImageList` está enlazado a un control secundario, no se podrán eliminar ni insertar imágenes en la mitad de la colección `ListImages`. A pesar de todo, se podrá agregar imágenes al final de la colección.

Una vez asociado un `ImageList` con un control común de Windows, se usará el valor de la propiedad `ImageIndex` o `ImageKey` para hacer referencia a un objeto `ListImage` en un procedimiento.

Puede encontrar más detalles sobre el uso de la clase `ImageList` en este enlace:

<https://learn.microsoft.com/es-es/dotnet/api/system.windows.forms.imagelist?view=netframework-4.8.1>

**1. Indique la opción correcta**

¿Cómo se hace referencia a un objeto ListItem en un procedimiento?:

Con propiedad ImageIndex o KeyImage

Con propiedad Index o Image

Con propiedad ImageIndex o ImageKey      X

**2. Indique la opción correcta**

¿A qué controles le puede proporcionar imágenes el control ImageList?:

ListView, TreeView      X

ComboBox, ListBox

Chart, DataGridView

**3. Indique la opción correcta**

¿Para qué sirve el control ImageList?:

Mostrar Imágenes

Contener Imágenes      X

Guardar Imágenes

**4. Indique la opción correcta**

El control TreeView se puede asociar al ImageList, pero debemos agregar los Nodos.

Verdadero      X

Falso

**5. Indique la opción correcta**

Se pueden agregar o eliminar imágenes a la lista una vez que ImageList está enlazado a un control secundario.

Verdadero      X

Falso

**6. Indique la opción correcta**

El control ImageList se utiliza independientemente de otros controles para ver imágenes.

Verdadero

Falso

X

## SP3/H2: Controles de presentación jerárquica de datos

Para resolver lo solicitado en la situación profesional, debemos tener en cuenta la utilización de controles avanzados de presentación de información. Este tipo de controles, además de facilitar la interpretación de la información en la interfaz, hará más ágil la carga o búsqueda de información para su posterior tratamiento.

**Control para visualización de la información en forma de árbol (TreeView)** En la situación profesional, podemos observar que debemos mostrar las provincias y departamentos en forma de árbol. Para hacer lo que nos solicitan, utilizaremos el control TreeView. El mismo está diseñado para mostrar datos de naturaleza jerárquica, como árboles organizativos, las entradas de un índice, o los archivos y directorios de un disco. Ejemplos:



Un control TreeView puede aplicarse para:

- Crear un árbol organizativo que pueda ser manipulado por el usuario.
- Crear un árbol que muestre, al menos, dos o más niveles de una base de datos.

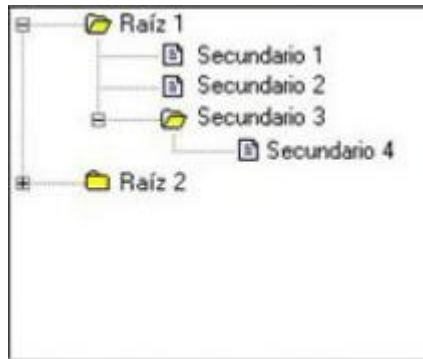
El control TreeView se usa a menudo junto con el control ListView. Esta combinación permite al usuario atravesar diferentes niveles jerárquicos: el control TreeView muestra la estructura de nivel superior, mientras que el control ListView muestra los conjuntos individuales de registros al seleccionar cada nodo. Siempre que se seleccione un nodo del control TreeView, el código llenará el control ListView con los conjuntos individuales de registros.

### Propiedades del control TreeView

Un "árbol" se compone de ramas sucesivas de "nodos". Cada nodo de la vista de árbol podría contener otros nodos, llamados secundarios. Puede mostrar nodos primarios o nodos que contienen nodos secundarios, como expandidos o contraídos. Un nodo puede expandirse o



contraerse dependiendo de si tiene o no nodos secundarios (nodos que parten de él). En el nivel superior están los nodos "raíz", y cada nodo raíz puede tener cualquier número de nodos secundarios. El número total de nodos no está limitado (salvo por las restricciones del sistema). En la siguiente figura se muestra un árbol con dos nodos raíz. "Raíz 1" tiene tres nodos secundarios y "Secundario 3" tiene, a su vez, un nodo secundario. "Raíz 2" tiene nodos secundarios, como indica el signo "+", pero está sin expandir.



También puede mostrar una vista de árbol con casillas de verificación junto a los nodos, estableciendo la propiedad CheckBoxes de la vista de árbol en true. De este modo, es posible activar o desactivar nodos mediante programación, determinando la propiedad Checked del nodo en true o false.



La apariencia del control TreeView se puede cambiar estableciendo algunas de sus propiedades de presentación y estilo. Si se establece ShowPlusMinus en true, se muestra un botón con el signo más (+) o con el signo menos (-), junto a cada nodo que se puede expandir o contraer, según corresponda.

Si se establece la propiedad ShowRootLines en true, el TreeView muestra líneas que unen entre sí todos los nodos de árbol raíz. Se pueden mostrar líneas que unan los nodos secundarios con su correspondiente nodo raíz, estableciendo la propiedad ShowLines en true.

Si se establece la propiedad HotTracking en true, cambia la apariencia de las etiquetas de los nodos cuando el puntero del mouse pasa sobre ellas, las etiquetas adquieren el aspecto de un

hipervínculo. También se puede personalizar totalmente la apariencia del control TreeView. Para ello, establezca la propiedad DrawMode en un valor distinto del Normal y controle el evento DrawNode.

Se pueden mostrar imágenes junto a los nodos del árbol asignando un ImageList a la propiedad ImageList del control y haciendo referencia al valor de índice de una Imagen de ImageList. Utilice las propiedades siguientes para asignar las imágenes:

- Establezca la propiedad ImageIndex en el valor de índice de la Imagen que desea que se muestre cuando no esté seleccionado un nodo del árbol.
- Determine la propiedad SelectedImageIndex en el valor de índice de la Imagen que desea que se muestre cuando esté seleccionado un nodo del árbol

### **Nodos del control TreeView**

La propiedad Nodes del control TreeView contiene la lista de nodos del nivel superior de la vista de árbol. Cada nodo de un árbol es en realidad un objeto TreeNode programable que pertenece a la colección TreeNodeCollection. Se puede acceder a esta colección a través de la propiedad Nodes del control TreeView. Como en otras colecciones, cada miembro de la colección tiene un valor único en las propiedades Index y Key, lo que permite el acceso a las propiedades del nodo.

Las imágenes a las que hacen referencia los valores de las propiedades ImageIndex y SelectedImageIndex del control Treeview son las imágenes predeterminadas que muestran todos los nodos de árbol asignados a la colección Nodes. Los nodos individuales del árbol pueden reemplazar las imágenes predeterminadas estableciendo las propiedades ImageIndex y SelectedImageIndex del nodo.

Los nodos de árbol se pueden expandir para mostrar el siguiente nivel de nodos secundarios. El usuario puede expandir también el nodo haciendo clic en el botón con el signo más (+), si se muestra al lado del nodo, o bien, llamando al método Expand.

Para expandir todos los niveles de nodos secundarios de un nodo primario, llame al método ExpandAll. El nivel secundario del nodo se puede contraer llamando al método Collapse, o presionando el botón con el signo menos (-) si se muestra al lado del nodo.

También se puede llamar al método Toggle para alternar los estados expandido y contraído del nodo, que cambiará del estado opuesto al actual, ya sea expandido o contraído.

### **Relaciones entre nodos**

Cada nodo del árbol puede ser secundario o primario, según su relación con otros nodos y tener propiedades que se pueden utilizar para explorar la vista del árbol:

**FirstNode:** obtiene el primer nodo secundario en la colección de nodos que se almacena en la propiedad Nodes del actual nodo. Si el nodo actual no tiene ningún nodo secundario, la propiedad FirstNode devolverá referencia de objeto null.

**LastNode:** obtiene el último nodo secundario en la colección de nodos que se almacena en la propiedad Nodes del actual nodo. Si éste no tiene ningún nodo secundario, la propiedad LastNode devolverá referencia de objeto null.

**NextNode:** obtiene el siguiente nodo relacionado en la colección de nodos que se almacena en la propiedad Nodes del nodo primario, del nodo actual. Si no hay ningún nodo a continuación, la propiedad NextNode devolverá referencia de objeto null.

**PrevNode:** obtiene el anterior nodo relacionado en la colección de nodos que se almacena en la propiedad Nodes del nodo primario, del nodo actual. Si no hay ningún nodo anterior, la propiedad PrevNode devolverá referencia de objeto null.

**Parent:** obtiene el nodo primario del actual nodo de árbol. Si el nodo está en el nivel de raíz, la propiedad Parent devolverá referencia de objeto null. El valor de la propiedad Parent es el nodo principal, del nodo actual. De existir los nodos secundarios del nodo actual, se enumerarán en su propiedad Nodes. El propio control TreeView dispone de la propiedad TopNode, que es el nodo raíz de toda la vista del árbol.

### **Para Agregar/Quitar nodos del árbol**

Puede agregar/quitar nodos a un control TreeView de tres maneras que describiremos a continuación:

#### **a) Utilizando la etiqueta inteligente asociada al control TreeView.**

Seleccione el control TreeView o agregue uno al formulario. Haga clic en el glifo de la etiqueta inteligente ( ) En el cuadro de diálogo Tareas de TreeView, seleccione Editar Nodos. En el Editor de la colección de nodos, haga clic en Agregar Raíz o Agregar Secundario para añadir un nodo primario o secundario, o presione el botón Quitar ( ) para eliminar nodos.

#### **b) Puede hacerlo desde la ventana de Propiedades:**

Seleccione el control TreeView o agregue uno al formulario. En la ventana de Propiedades, haga clic en el botón de puntos suspensivos ( ) situado junto a la propiedad Nodes. En el Editor de la colección de nodos, haga clic en Agregar Raíz o Agregar

Secundario para añadir un nodo primario o secundario, o presione el botón Quitar ( ) para eliminar nodos.

**c) Puede hacerlo utilizando código (o programación):**

Para agregar un nodo utilizando código debe utilizar el método Add de la propiedad Nodes de la vista de árbol. Este método agrega el nodo al final de la colección:

- `TreeView.Nodes.Add(String)`: agrega un nuevo nodo con el texto de etiqueta especificado.
- `TreeView.Nodes.Add(Nodo)`: agrega un nodo anteriormente creado. - `TreeView.Nodes.Add(String, String)`: crea un nuevo nodo con la clave y texto especificados.
- `TreeView.Nodes.Add(String, String, Int32)`: crea un nuevo nodo con la clave, texto e imagen(índice) especificados.
- `TreeView.Nodes.Add(String, String, String)`: crea un nuevo nodo con la clave, texto e imagen(clave) especificados.
- `TreeView.Nodes.Add(String, String, Int32, Int32)`: crea un nuevo nodo con la clave, texto, imagen(índice) cuando no está seleccionado, imagen(índice) cuando está seleccionado.
- `TreeView.Nodes.Add(String, String, String, String)`: crea un nuevo nodo con la clave, texto, imagen(clave) cuando no está seleccionado, imagen(clave) cuando está seleccionado. El ejemplo de código siguiente requiere que tenga un formulario con un control TreeView ya agregado:

```
TreeNode nodo = new TreeNode("Texto del nodo");  
treeView1.Nodes.Add(nodo);
```

Para quitar un nodo mediante programación debe utilizar el método Remove de la propiedad Nodes de la vista del árbol para quitar un solo nodo, o el método Clear para borrar todos los nodos.

El ejemplo de código siguiente requiere que tenga un formulario con un control TreeView ya agregado:

```
treeView1.Nodes[0].Remove();  
  
treeView1.Nodes.Clear();
```

En el primer ejemplo se elimina el primer nodo del treeview, si ese nodo tuviera nodos hijos éstos también se borrarán completamente.

En el segundo caso se eliminan todos los nodos del control treeview (método Clear).

## Recorrer todos los nodos del árbol

Para realizar esta actividad debemos:

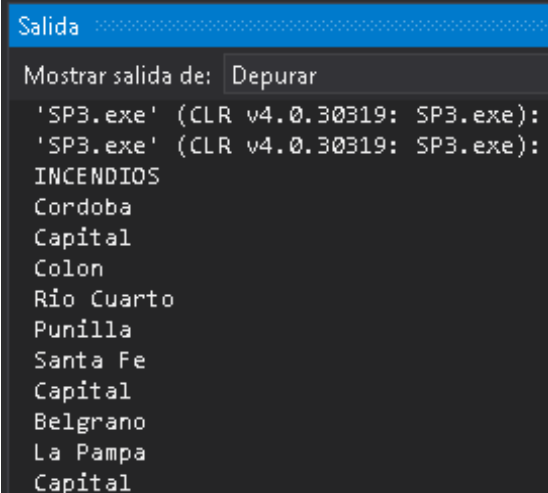
1. Crear un procedimiento recursivo que compruebe cada nodo.
2. Llamar al procedimiento.

A continuación, se muestra cómo visualizar la propiedad Text de cada nodo:

```
private void ListarNodos()
{
    foreach(TreeNode nodo in treeview1.Nodes)
    {
        // comienza con los nodos raiz del treeview
        RecorrerRecursivo(nodo);
    }
}

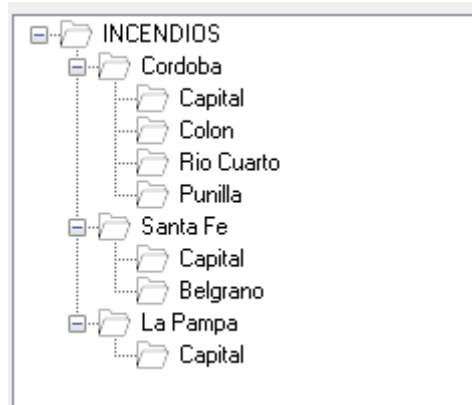
private void RecorrerRecursivo(TreeNode nodo)
{
    Trace.WriteLine(nodo.Text); // muestra en la consola el texto del nodo
    foreach(TreeNode n in nodo.Nodes)
    {
        RecorrerRecursivo(n); // repite para cada nodo hijo
    }
}
```

La salida por consola muestra:



```
Salida
Mostrar salida de: Depurar
'SP3.exe' (CLR v4.0.30319: SP3.exe):
'SP3.exe' (CLR v4.0.30319: SP3.exe):
INCENDIOS
Cordoba
Capital
Colon
Rio Cuarto
Punilla
Santa Fe
Capital
Belgrano
La Pampa
Capital
```

Que son los textos de estos nodos cargados en el treeview:



### Los eventos del control TreeView son:

**AfterSelect:** se produce después de seleccionarse el nodo, es el evento predefinido.

**AfterCheck:** se produce después de activarse la casilla de verificación del nodo.

**AfterCollapse:** se produce después de contraerse el nodo.

**AfterExpand:** se produce después de expandirse el nodo.

### ¿Cómo determinar en qué nodo del TreeView se hizo clic?

Cuando se trabaja con el control TreeView, una tarea frecuente es determinar en qué nodo se hizo clic, para lo cual trabajaremos con el evento AfterXxxx mencionados en el punto anterior, por ejemplo, **AfterSelect**: se debe comprobar la clase TreeViewEventArgs, que contiene datos relacionados con el evento provocado por el usuario. La propiedad Node obtiene el nodo que se ha comprobado, expandido, contraído o seleccionado.

En el siguiente ejemplo, el controlador de eventos recibe un argumento de tipo TreeViewEventArgs, que contiene datos relacionados con este evento. Luego, se utiliza la propiedad Node, que contiene una referencia al nodo cliqueado:

```
private void treeView1_AfterSelect(object sender, TreeViewEventArgs e)
{
    MessageBox.Show(e.Node.Text); // muestra el texto del nodo que se seleccionó
}
```

Puede consultar más información en este enlace:

<https://learn.microsoft.com/es-es/dotnet/api/system.windows.forms.treeview?view=netframework-4.8.1>

## SP3/Autoevaluación 2

1. **Indique la opción correcta.** Los nodos se pueden expandir o contraer sólo por la acción del usuario.

Verdadero

Falso            X

2. **Indique la opción correcta.** La propiedad Nodes permite acceder a los nodos del árbol.

Verdadero      X

Falso

3. **Indique la opción correcta.** Solo se pueden agregar o quitar nodos utilizando el editor de la colección de nodos.

Verdadero

Falso            X

4. **Indique la opción correcta.** ¿Cuál es la aplicación de un TreeView?

- Mostrar información aleatoria
- Mostrar información en orden jerárquico    X
- Mostrar información en orden
- Mostrar información de base de datos

5. **Indique la opción correcta.** La propiedad Parent obtiene el nodo primario del nodo actual.

Verdadero

Falso            X

## SP3/H3: Controles para visualización de información en forma de lista (ListView)

El control **ListView** muestra una lista de elementos con íconos. Este control es idóneo para representar subconjuntos de datos (como los miembros de una base de datos) u objetos discretos (como plantillas de documentos).

Un control ListView puede aplicarse:

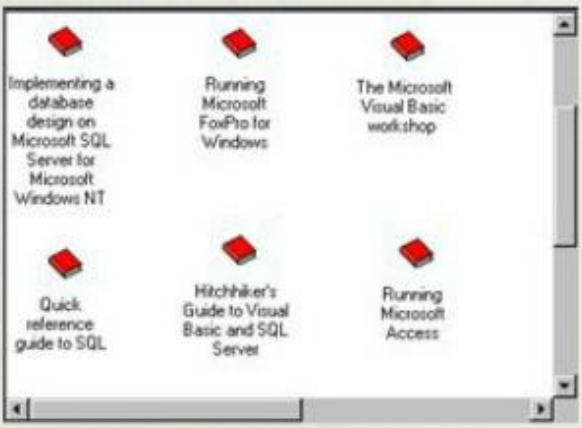


A mostrar el resultado de una consulta de una base de datos.

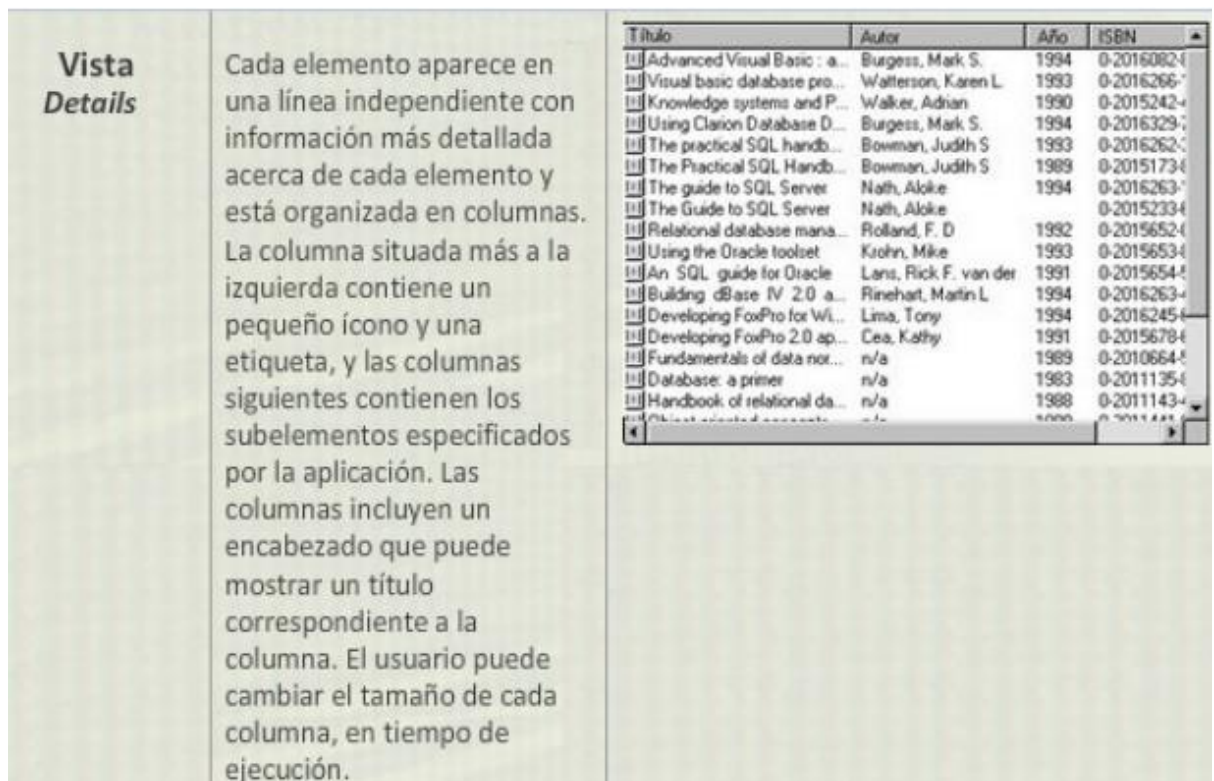
A mostrar todos los registros de una tabla de una base de datos.

Junto con un control TreeView; y dar al usuario una visión ampliada de la información relacionada a un nodo del control TreeView.

El control tiene cuatro modos de vista: LargeIcon, SmallIcon, List y Details. Cada uno de ellos tiene una ventaja en particular con respecto a los demás. Algunos se indican en la tabla siguiente:



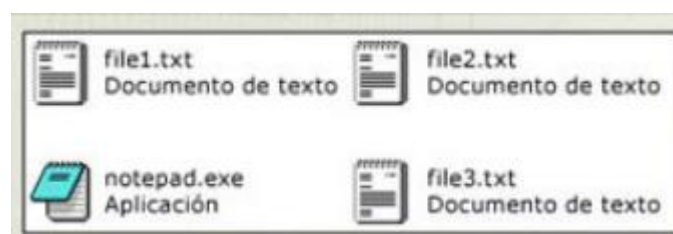
Modo de	Ventaja	Imagen
<b>Vista Largedcon</b>	Cada elemento aparece como un ícono de tamaño normal debajo del cual figura una etiqueta.	
<b>Vista SmallIcon</b>	Cada elemento aparece como un ícono pequeño a cuya derecha figura una etiqueta.	
<b>Vista List</b>	Cada elemento aparece como un ícono pequeño a cuya derecha figura una etiqueta. Los elementos están organizados en columnas sin encabezado.	



Un modo de vista adicional, Mosaico. La característica de la vista en mosaico del control ListView permite ofrecer un equilibrio visual entre la información gráfica y la textual.

La información textual mostrada de un elemento en la vista en mosaico es igual que la información de columna definida para la vista de detalles.

La vista en mosaico utiliza un ícono de 32 x 32 píxeles y varias líneas de texto, como se muestra en las siguientes imágenes:



## Mostrar Íconos

El control ListView puede mostrar íconos procedentes de tres listas de imágenes:

Las vistas List, Details y SmallIcon muestran imágenes procedentes de la lista de imágenes especificada en la propiedad SmallImageList.

La vista LargeIcon muestra imágenes procedentes de la lista de imágenes especificada en la

propiedad `LargeImageList`.

Asimismo, una vista de lista puede mostrar un conjunto adicional de íconos, que se establece en la propiedad `StateImageList`, junto a los íconos grandes o pequeños.

### **Para Agregar/Quitar elementos**

Puede agregar/quitar elementos a un control `ListView` de tres maneras:

#### **a) Puede utilizar la etiqueta inteligente asociada al control `ListView`:**

Seleccione el control `ListView` o agregue uno al formulario. Haga clic en el glifo de la etiqueta inteligente ( ).

En el cuadro de diálogo Tareas de `ListView`, seleccione `Editar Elementos`.

En el Editor de la colección de `Items`, haga clic en `Agregar` o `Quitar`, para agregar y quitar ítems.

#### **b) Puede utilizar la ventana de Propiedades:**

Seleccione el control `ListView` o agregue uno al formulario.

En la ventana de `Propiedades`, haga clic en el botón de puntos suspensivos ( ) situado junto a la propiedad `Items`.

En el Editor de la colección de `Items`, haga clic en `Agregar` o `Quitar`, para agregar y quitar ítems.

#### **c) Puede utilizar código:**

El proceso para agregar un elemento a un control `ListView` consiste básicamente en especificar el elemento y asignarle propiedades.

Utilice el método `Add` de la propiedad `Items` del control `ListView`:

`ListView.Items.Add(String)`: agrega un nuevo elemento con el texto especificado.

`ListView.Items.Add(Item)`: agrega un elemento anteriormente creado.

`ListView.Items.Add(String, String)`: crea un nuevo elemento con el texto y la imagen(clave) especificados.

`ListView.Items.Add(String, Int32)`: crea un nuevo elemento con el texto e imagen(índice) especificados.

`ListView.Items.Add(String, String, String)`: crea un nuevo elemento con la clave, texto e imagen(clave) especificados.

`ListView.Items.Add(String, String, Int32)`: crea un nuevo elemento con la clave, texto e imagen(índice) especificados.

### **Quitar elementos**

Para quitar elementos mediante programación utilice el método RemoveAt o Clear de la propiedad Items. El método RemoveAt quita un solo elemento; el método Clear quita todos los elementos de la lista.

### Para mostrar imágenes en una vista de lista

Será necesario contar con un control de tipo ImageList cargado con algunas imágenes o iconos para usar como repositorio desde el cuál se tomarán las imágenes para mostrar sobre el control ListView.


1. Establezca la propiedad adecuada (SmallImageList, LargeImageList o StateImageList) en el componente ImageList existente que desee utilizar.
2. Establezca la propiedad ImageIndex o StateImageIndex para cada elemento de la lista que tenga un ícono asociado.

```
ListView1.SmallImageList = ImageList1;  
ListView1.Items(0).ImageIndex = 3;
```

### Para agregar/quitar columnas en una vista de detalle

En la vista Detalles, el control ListView puede mostrar varias columnas para cada elemento de la lista. Puede utilizar las columnas para mostrar al usuario información de diversos tipos respecto de cada elemento de la lista.

Por ejemplo, una lista de libros puede mostrar el título, el autor, el año, etc.



Título	Autor	Año	ISBN
dBASE IV : complete reference for...	Hergert, Douglas.	1989	1-5561!
dBASE IV : programmer's quick ref...	Viescas, John	1989	1-5561!
Hitchhiker's Guide to Visual Basic ...	Vaughn, William R.	1996	1-5561!

Puede agregar/quitar columnas a un control ListView de varias maneras. Utilizando la etiqueta inteligente asociada al control ListView, o quizás desde la ventana de Propiedades. También puede agregar/quitar columnas utilizando un código.

#### a) Utilizando la ventana de propiedades:

Seleccione el control ListView o agregue uno al formulario. En la ventana de Propiedades, haga clic en el botón de puntos suspensivos ( ) situado junto a la propiedad Columns. En el Editor de la colección de Columnas haga clic en Agregar o Quitar, para agregar y quitar columnas.

**b) Mediante la etiqueta inteligente:**

Seleccione el control ListView o agregue uno al formulario. Haga clic en el glifo de la etiqueta inteligente ( ). En el cuadro de diálogo Tareas de ListView, seleccione Editar Columnas. En el Editor de la colección de Columnas haga clic en Agregar o Quitar, para agrega y quitar columnas.

**c) Mediante programación:**

El proceso de agregar una columna a un control ListView consiste básicamente en especificar la columna y asignarle propiedades. Utilice el método Add de la propiedad Columns del control ListView:

- ListView.Columns.Add(String): agrega una nueva columna con el texto especificado.
- ListView.Columns.Add(Columna): agrega una columna anteriormente creada.
- ListView.Columns.Add(String, String): crea una nueva columna con la clave y texto especificados.
- ListView.Columns.Add(String, Int32): crea una nueva columna con el texto y ancho especificados.
- ListView.Columns.Add(String, Int32, HorizontalAlignment): crea una nueva columna con el texto, ancho y valor de HorizontalAlignment.
- ListView.Columns.Add(String, String, Int32): crea una nueva columna con la clave, texto y ancho especificados.
- ListView.Columns.Add(String, String, Int32, HorizontalAlignment, Int32): crea una nueva columna con la clave, texto, ancho, valor de HorizontalAlignment y valor de índice de la imagen que se va a mostrar en la columna.
- ListView.Columns.Add(String, String, Int32, HorizontalAlignment, String): crea una nueva columna con la clave, texto, ancho, valor de HorizontalAlignment y valor clave de la imagen que se va a mostrar en la columna.

Los valores posibles de HorizontalAlignment son:

- Center: el texto se alinea en el centro.
- Left: el texto se alinea a la izquierda.

- Right: el texto se alinea a la derecha.

### Para mostrar subelementos en las columnas

El control ListView puede mostrar texto adicional, o subelementos, para cada elemento de la vista Details. La primera columna muestra el texto del elemento; por ejemplo, el número del empleado. La segunda, tercera y siguientes columnas muestran el primero, segundo y siguientes subelementos asociados.

Para agregar subelementos a un elemento de la lista:

1. Agregue las columnas necesarias. Como la primera columna mostrará la propiedad Text del elemento, necesitará una columna más por cada subelemento.
2. Llame al método Add de la colección que devuelve la propiedad SubÍtems de un elemento:
  - ListView.Items(indice).SubItems.Add(SubItem): agrega un subítem creado anteriormente.
  - ListView.Items(indice).SubItems.Add(String): agrega un subítem con el texto especificado.
  - ListView.Items(indice).SubItems.Add(string, Color, Color, Font): agrega un subítem con el texto, color del primer plano, del fondo y el tipo de letra del subelemento.

El siguiente ejemplo establece el nombre de una calle y su numeración, para un elemento de la lista:

```
ListView1.Items(0).SubItems.Add("Calle Rondeau");  
ListView1.Items(0).SubItems.Add("165");
```

Recuerde que la primera columna del control ListView muestra la propiedad text del Item. El resto de las columnas corresponden a los subítems.

Más detalles sobre esta clase:

<https://learn.microsoft.com/es-es/dotnet/api/system.windows.forms.listview?view=netframework-4.8.1>

**1. Indique la opción correcta**

El siguiente código limpia un control ListView: `ListView1.Items.Clear();`

Verdadero                      X

Falso

**2. Indique la opción correcta**

¿Que tipo de vista se asigna para que el control ListView muestre las columnas?

SmallIcon

LargeIcon

Details                      X

List

**3. Indique la opción correcta**

¿Cuál es la aplicación de un ListView?

Junto con un control TreeView; y dar al usuario una visión ampliada de un nodo del control TreeView.                      X

Junto con un control ComboBox; y dar al usuario una visión ampliada de un nodo del control.

Junto con un control ListBox; y dar al usuario una visión ampliada de un nodo del control ListView.

**4. Indique la opción correcta**

¿Cuál el el tipo de vista que tiene el control ListView para íconos pequeños?

SmallIcon                      X

LargeIcon

Details

List

**5. Indique la opción correcta**

Para borrar un elemento de un listView debemos ingresar:

`ListView1.Items.Add("Elemento-3", 3)`

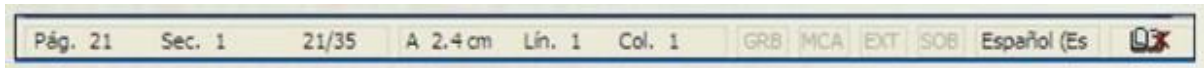
Verdadero

Falso            X



## SP3/H4: Control StatusStrip, barras de estado

Los controles StatusStrip muestran información sobre objetos que se van a visualizar en una ventana o cuadro de diálogo: los componentes del objeto o información contextual respecto a la operación del objeto dentro de la aplicación.



Un control StatusStrip puede aplicarse a:

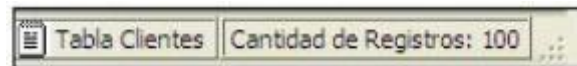
- Informar al usuario sobre las medidas de una tabla de una base de datos, por ejemplo, el número de registros y la posición actual en la base de datos.
- Indicar el estado de ciertas teclas (por ejemplo, bloq mayús o bloq núm).

Las propiedades más importantes que podemos mencionar del control StatusStrip son:

<i>BackColor</i>	obtiene o establece el color de fondo del control.
<i>BackgroundImage</i>	obtiene o establece la imagen de fondo que se muestra en el control.
<i>Dock</i>	obtiene o establece que los bordes del control se acoplarán a su control principal (generalmente un formulario), y determina cómo se cambia el tamaño de un StatusStrip, con su elemento primario.
<i>Font</i>	obtiene o establece la fuente utilizada para mostrar texto en el control.
<i>Items</i>	obtiene todos los elementos que puede contener el objeto StatusStrip.
<i>ShowItemToolTips</i>	obtiene o establece un valor que indica si se muestra información sobre herramientas para StatusStrip.

Normalmente, un control StatusStrip está compuesto de objetos ToolStripStatusLabel, que muestran textos, íconos o ambos.

El control ToolStripStatusLabel representa el panel de un control StatusStrip. Puede contener texto o un ícono que refleja el estado de una aplicación.



Las propiedades más importantes que podemos mencionar de este control son:

<i>BorderSides</i>	obtiene o establece un valor que indica qué lados del control ToolStripStatusLabel tienen bordes.
<i>BorderStyle</i>	obtiene o establece el estilo de los bordes del control.
<i>DisplayStyle</i>	obtiene o establece si se muestran el texto y las imágenes en un ToolStripItem.
<i>Font</i>	obtiene o establece la fuente del texto que muestra el elemento.
<i>Image</i>	obtiene o establece la imagen que se muestra en un control ToolStripItem.
<i>Text</i>	obtiene o establece el texto que se mostrará en el elemento.
<i>TextAlign</i>	obtiene o establece la alineación del texto en un ToolStripLabel.
<i>ToolTipText</i>	obtiene o establece el texto que aparece como ToolTip (pequeña ventana emergente rectangular que muestra una breve descripción de la finalidad de un control cuando el usuario sitúa el puntero del mouse sobre el control).

El control StatusStrip también puede contener los controles:

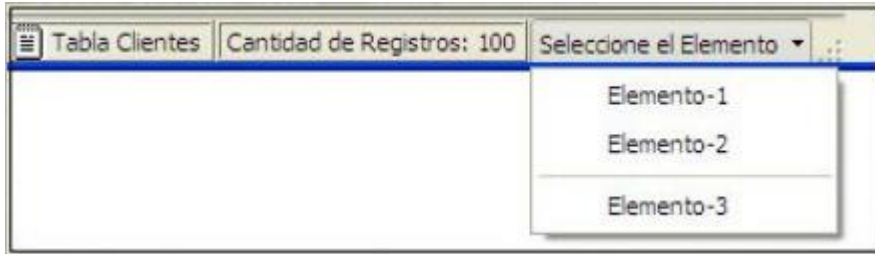
- a) ToolStripDropDownButton,
- b) ToolStripSplitButton y

c) ToolStripProgressBar.

**a) Control ToolStripDropDownButton**

Representa un control en el que, al hacer clic el usuario puede seleccionar un elemento único.

Utilice ToolStripDropDownButton para activar controles desplegables familiares, como los selectores de color. Es similar a un menú contextual de Windows.

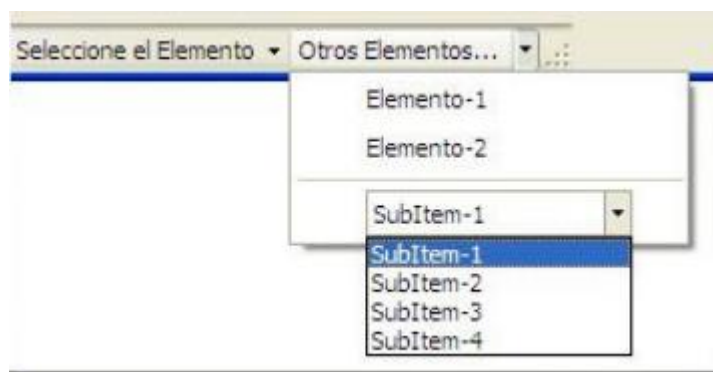


Las propiedades más importantes que podemos mencionar de este control son:

<i>DisplayStyle</i>	muestran el texto y las imágenes en un ToolStripItem.
<i>DropDownItems</i>	obtiene la colección de elementos que puede contener el objeto ToolStripDropDownButton. Utilice el editor de esta colección para agregar/quitar elementos y configurar propiedades. Los elementos que se pueden agregar a un control ToolStripDropDownButton son: • MenuItem; • ComboBox; • Separador; • TextBox.
<i>Font</i>	determina la fuente del texto que muestra el elemento.
<i>Image</i>	obtiene la imagen que se muestra en un control ToolStripItem.
<i>Text</i>	establece el texto que se mostrará en el elemento.
<i>TextAlign</i>	facilita la alineación del texto.
<i>ToolTipText</i>	permite la aparición del texto, que aparece como ToolTip (pequeña ventana emergente rectangular que muestra una breve descripción de la finalidad de un control cuando el usuario sitúa el puntero del mouse sobre el control).

#### b) Control ToolStripSplitButton

Representa una combinación de un botón estándar situado a la izquierda y un botón de lista desplegable situado a la derecha, o la combinación contraria si el valor de RightToLeft es Yes.



Las propiedades más importantes que podemos mencionar de este control son las mismas que hemos mencionado para el control `ToolStripDropDownButton`.

### c) Control `ToolStripProgressBar`

Representa un control de barra de progreso de Windows. Indica visualmente el progreso de una operación larga. El control `ToolStripProgressBar` muestra una barra que se llena de izquierda a derecha con el color de resaltado del sistema a medida que progresa la operación.



Las propiedades **Maximum** y **Minimum** definen el intervalo de valores que representarán el progreso de una tarea. Normalmente, la propiedad **Minimum** se establece en cero, y la propiedad **Maximum** se establece en un valor que indica que la tarea ha terminado. Por ejemplo, para mostrar correctamente el progreso al copiar un grupo de archivos, la propiedad **Maximum** se podría establecer en el número total de archivos que se van a copiar. La propiedad **Value** representa el progreso que la aplicación realiza para terminar la operación.

### Para agregar/quitar controles a la barra de estado

Puede agregar/quitar controles a la barra de estado de las mismas tres maneras que los controles anteriores:

#### a) Utilizando la ventana de Propiedades:

Seleccione el control `StatusStrip` o agregue uno al formulario. En la ventana de Propiedades, haga clic en el botón de puntos suspensivos ( `...` ) situado junto a la propiedad `Items`. En el Editor de la colección de `Items`, haga clic en `Agregar` o `Quitar` para agregar y quitar controles.

### b) Utilizando la etiqueta inteligente:

Seleccione el control StatusStrip o agregue uno al formulario. Haga clic en el glifo de la etiqueta inteligente ( ). En el cuadro de diálogo Tareas de StatusStrip, seleccione Editar Elementos. En el Editor de la colección de Items, haga clic en Agregar o Quitar para agregar y quitar controles.

### c) Utilizando programación:

El proceso de agregar un control a una barra de estado consiste básicamente en especificar el elemento y en asignarle propiedades. Utilice el método Add de la propiedad Items del control StatusStrip:

- StatusStrip.Items.Add(Image): agrega ToolStripItem que muestra la imagen especificada a la colección.
- StatusStrip.Items.Add(String): agrega un ToolStripItem que muestra el texto especificado.
- StatusStrip.Items.Add(Item): agrega el elemento especificado al final de la colección.
- StatusStrip.Items.Add(String, Imagen): agrega un ToolStripItem que muestra el texto y la imagen especificados.
- StatusStrip.Items.Add(String, Imagen,EventHandler): agrega un ToolStripItem que muestra el texto, y la imagen especificados a la colección y provoca el evento Click.

El ejemplo de código siguiente requiere que tenga un formulario con un control StatusStrip agregado:

```
ToolStripProgressBar barra = new ToolStripProgressBar();  
sspEstado.Items.Add(barra);  
barra.Maximum = 100;
```

También se puede utilizar el método AddRange para agregar un control ToolStripStatusLabel a la barra de estado.

### Quitar controles

Para quitar controles mediante programación utilice el método **RemoveAt** o **Clear** de la propiedad Items. El método RemoveAt quita un solo control; el método Clear quita todos los controles de la barra de estado.

Consulte más detalles sobre la clase en el siguiente enlace:

<https://learn.microsoft.com/es-es/dotnet/desktop/winforms/controls/statusstrip-control-overview?view=netframeworkdesktop-4.8>

Retomando los requerimientos de la situación profesional podemos concluir que:

- Un control TreeView nos permite mostrar en forma jerárquica los datos de las provincias y sus departamentos.
- Un control ListView resuelve de forma eficiente el manejo de la información sobre los tipos de incendio y cantidades referidos al nodo seleccionado en el control TreeView
- Agregar un control StatusStrip permitirá contar con un componente apto para mostrar los totales de incendios requeridos en cada caso.

**1. Indique la opción correcta.**

Un control StatusStrip puede contener los siguientes controles: ToolStripStatusLabel, DropDownButton, SplitButton y ToolStripProgressBar.

Verdadero

Falso            X

**2. Indique la opción correcta**

¿Para qué se usa el control StatusStrip?

Se utiliza en conjunto a un control MenuStripControl.

Informar al usuario sobre las medidas de una tabla de una base de datos, por ejemplo, el número de registros y la posición actual en la base de datos.            X

Informar al usuario sobre las medidas de una tabla de una base de datos.

**3. Indique la opción correcta**

¿Cuál es la propiedad que permite obtener los elementos contenidos en un Label en el control StatusStrip?

ShowItemToolTips

ToolStripStatusLabel            X

BorderSides

Text

**4. Indique la opción correcta**

¿Qué control contiene un StatusStrip?

ComboBox

ListBox

ProgressBar    X

GridView

**5. Indique la opción correcta**

Para agregar o quitar controles de un StatusBar debemos utilizar, en el modo de diseño, los botones.



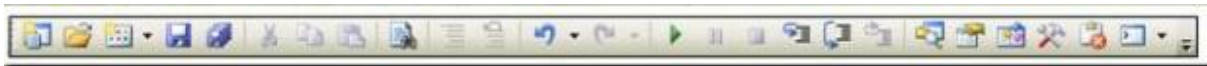
Verdadero

Falso            X

## SP3/H5: Control ToolStrip, barras de herramientas

Si bien en la situación profesional planteada no se requiere el uso de una barra de herramientas, conocida como control ToolStrip en Visual Studio, creemos conveniente dedicarle un espacio para tomar contacto con este control que puede ser de mucha utilidad en aplicaciones similares a la planteada acá.

El control ToolStrip permite crear barras de herramientas que se pueden asociar a una aplicación. Por ejemplo, la siguiente imagen:



Normalmente, una barra de herramientas contiene botones que corresponden a elementos de algún menú de la aplicación, lo que proporciona una interfaz gráfica para que el usuario tenga un acceso rápido a las funciones y comandos utilizados con más frecuencia.

Las propiedades más importantes que podemos mencionar del control **ToolStrip** son:

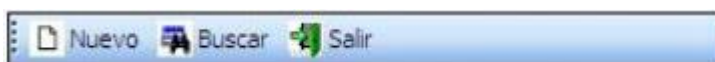
- **BackColor**: obtiene o establece el color de fondo del ToolStrip.
- **Dock**: obtiene o establece que los bordes del ToolStrip se acoplarán a su control principal y determina cómo se cambia el tamaño de un ToolStrip con su elemento primario.
- **Font**: obtiene o establece la fuente utilizada para mostrar texto en el control.
- **Items**: obtiene todos los elementos que contiene el ToolStrip. Esta propiedad será tratada en profundidad más adelante.
- **LayoutStyle**: obtiene o establece un valor que indica cómo se disponen los elementos en la barra de herramientas.
- **ShowItemToolTips**: obtiene o establece un valor que indica si se mostrará información sobre herramientas en los elementos de ToolStrip

### Elementos de un control ToolStrip

El control ToolStrip puede contener los siguientes controles:

#### a) Control **ToolStripButton**

Representa un botón de barra de herramientas que admite texto e imagen.



Las propiedades más importantes que podemos mencionar de este control son:

- **DisplayStyle:** obtiene o establece si se muestran el texto y las imágenes en un ToolStripButton.
- **Image:** obtiene o establece la imagen que se muestra en el botón.
- **ImageAlign:** obtiene o establece la posición de la imagen.
- **Text:** obtiene o establece el texto que se mostrará en el botón.
- **TextAlign:** obtiene o establece la posición del texto.
- **ToolTipText:** obtiene o establece el texto que aparece como ToolTip (pequeña ventana emergente rectangular que muestra una breve descripción de la finalidad de un control, cuando el usuario sitúa el puntero del mouse sobre el mismo).

Generalmente se utiliza el evento “Click” de este control para escribir las acciones que se deben ejecutar cuando el usuario presiona un botón de la barra de herramientas. Es similar al comportamiento de los botones de comando ampliamente usados (Control Button).

#### b) Control **ToolStripComboBox**

Este control muestra un campo de edición combinado con un ListBox y permite al usuario seleccionar una opción de la lista o escribir texto nuevo.

De forma predeterminada, un ToolStripComboBox muestra un campo de edición con una lista desplegable oculta.

La propiedad DropDownStyle determina el estilo que mostrará el cuadro combinado.

#### c) Control **ToolStripSplitButton**

Representa una combinación de un botón estándar situado a la izquierda y un botón de lista desplegable situado a la derecha, o la combinación contraria si el valor de la propiedad RightToLeft es Yes.

#### d) Control **ToolStripLabel**

Muestra una etiqueta de texto que suele utilizarse en una barra de estado o en un ToolStrip como un comentario o título.

#### e) Control **ToolStripSeparator**

Representa una línea utilizada para agrupar elementos de ToolStrip.

#### f) Control **ToolStripDropDownButton**

Representa un control en el que, al hacer clic, muestra una lista asociada donde el usuario puede seleccionar un elemento único.

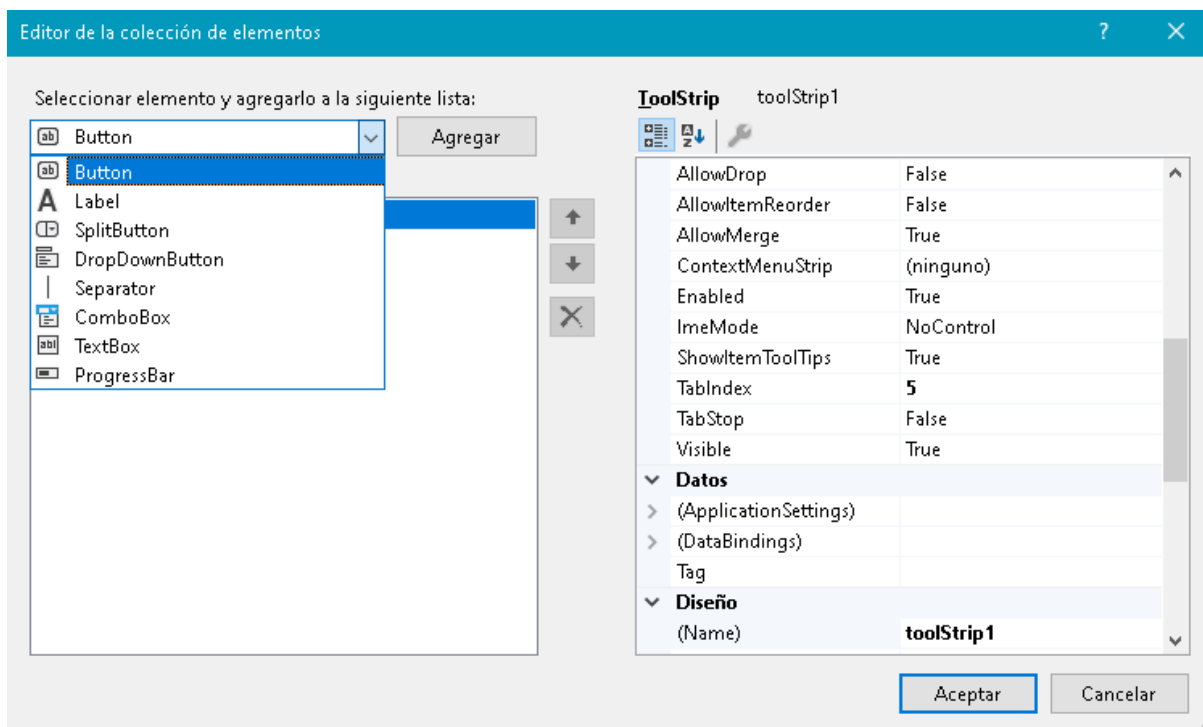
g) Control **ToolStripProgressBar**

Representa un control de barra de progreso de Windows.

h) Control **ToolStripTextBox**

Representa un cuadro de texto en un control ToolStrip que permite al usuario escribir texto.

Podemos editar el contenido del control ToolStrip desde la ventana del editor de elementos:



Podemos ver la lista de opciones disponibles para agregar en la lista desplegable de la izquierda, a la derecha tenemos las propiedades del elemento seleccionado.

Podrá encontrar más información y referencias de este control en este enlace:

<https://learn.microsoft.com/en-us/dotnet/api/system.windows.forms.toolstrip?view=netframework-4.8.1>

**1. Indique la opción correcta**

¿Cómo se quitan controles a la barra de herramientas?

Remove

Clear

RemoveAt     X

Delete

**2. Indique la opción correcta**

El control ToolStripComboBox se utiliza para listas desplegables.

Verdadero     X

Falso

**3. Indique la opción correcta**

¿Qué controles puede contener un ToolStrip?

ComboBox

ToolStripButton     X

ProgressBar

GridView

**4. Indique la opción correcta**

Para agregar elementos en un control ToolStrip se puede usar ToolStrip1.Items.Add()

Verdadero     X

Falso

**5. Indique la opción correcta**

El control ToolStripSeparator se utiliza para agrupar controles.

Verdadero     X

Falso

**6. Indique la opción correcta**

¿Para qué se usa el control ToolStrip?

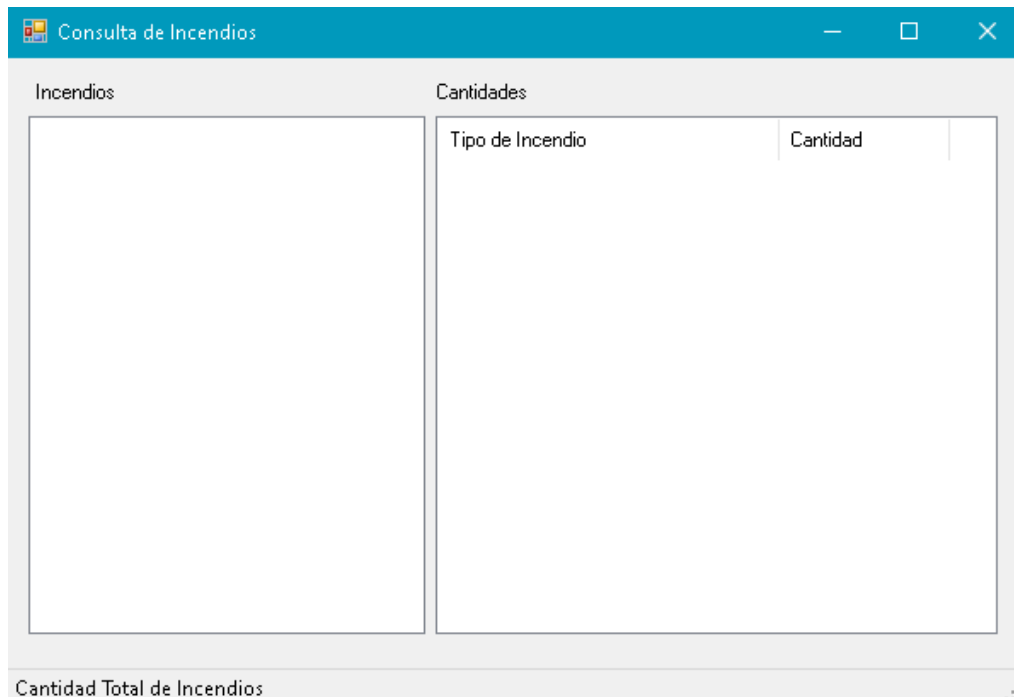
Se utiliza en conjunto a un control StatusStripControl.

Informar al usuario sobre las medidas de una tabla de una base de datos, indicando, por ejemplo, el número de registros y la posición actual en la base de datos.

Acceso rápido a las funciones y comandos utilizados con más frecuencia. ☒ X

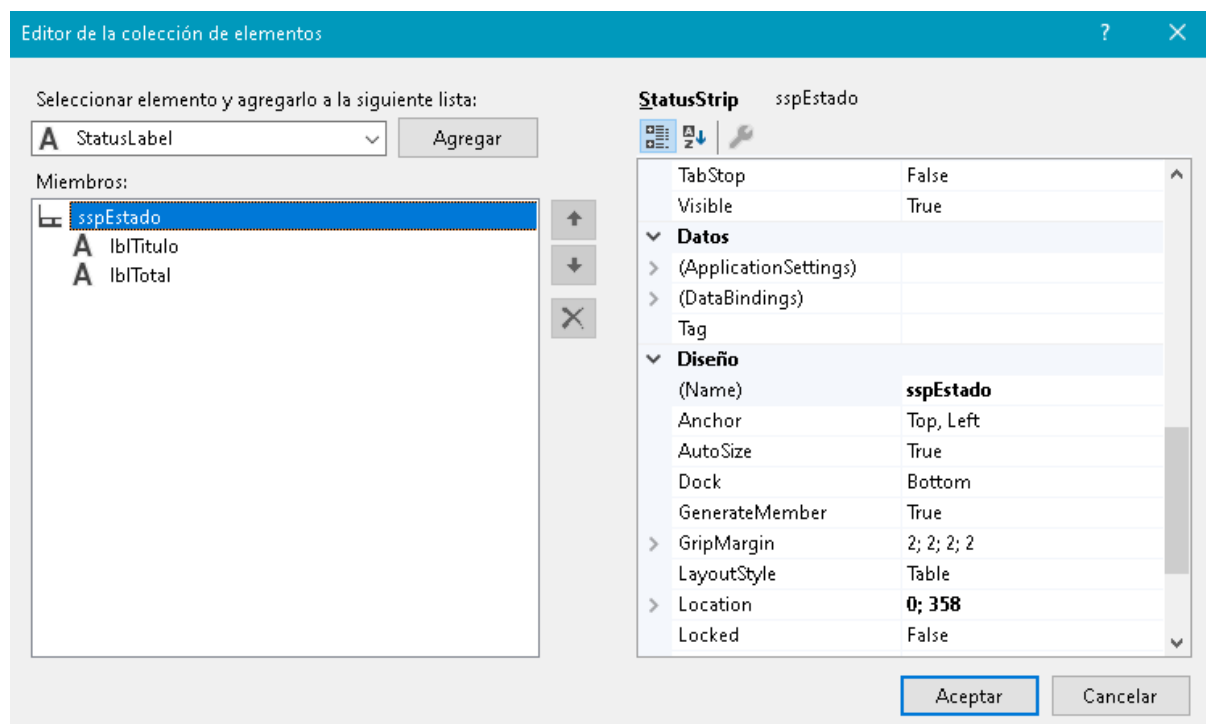
## SP3/Ejercicio resuelto

Para esta resolución crearemos un nuevo proyecto de Visual Studio y procederemos a diseñar y configurar todos los componentes del formulario que compone la interfaz gráfica.



Se necesitan 2 “Labels”, un control “TreeView” (tvwIncendios), un control “ListView” (lvwCantidades) y un control “StatusStrip” (sspEstado).

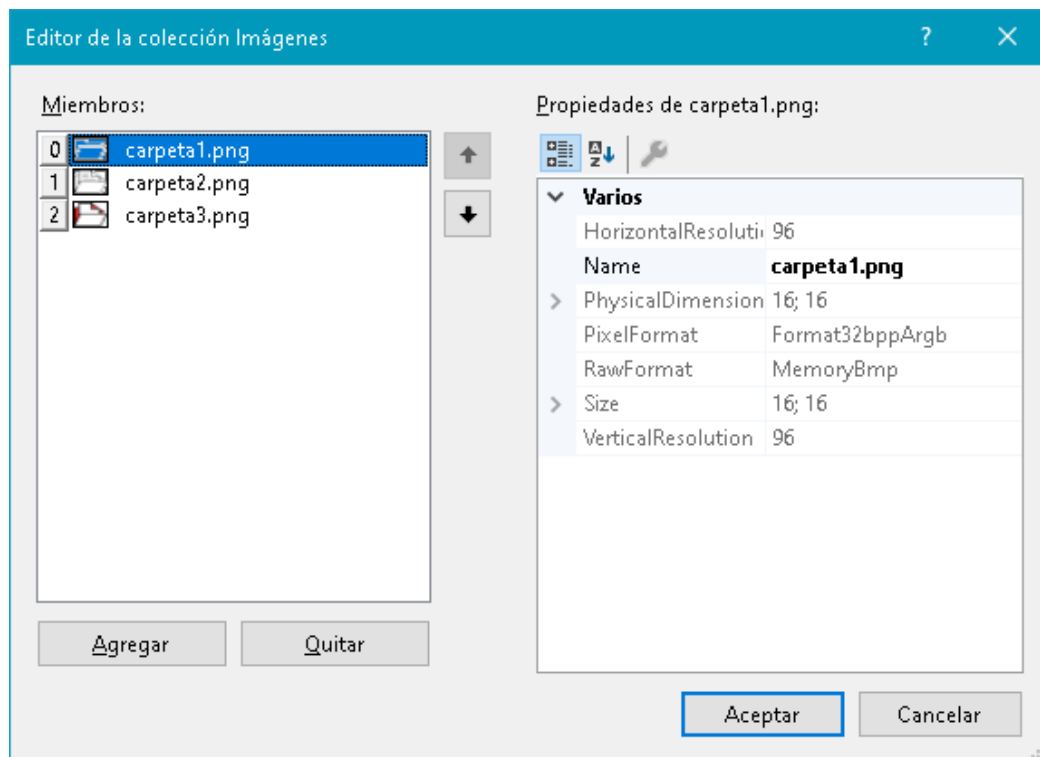
En el control “StatusStrip” agregamos dos componentes de tipo “StatusLabel”:



En “lblTitulo” asignaremos a la propiedad “Text” el valor “Cantidad Total de Incendios”.

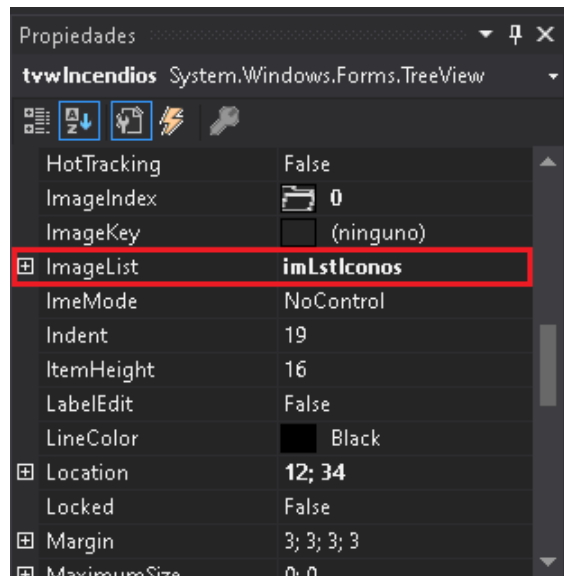
En “lblTotal”, la propiedad “Text” quedará vacía, su valor será asignado por código dependiendo del nodo que el usuario seleccione en el TreeView.

Para completar la presentación de los datos en el TreeView vamos a agregar al formulario un control “ImageList”, su nombre será “imLstIconos”. Seguidamente configuraremos sus elementos con 3 archivos de tipo “bmp”, “jpg” o “png”, estos archivos de iconos se usarán para asignar a cada tipo de nodo en el treeview.



Una vez cargadas las imágenes en el control ImageList debemos vincular el TreeView para que tome este control como fuente para los iconos de los nodos, eso se logra simplemente asignado el nombre del control ImageList “imLstIconos” a la propiedad “ImageList” del TreeView:





De esta forma queda completo el diseño y configuración del formulario, continuaremos ahora con la implementación de las clases que necesitamos construir para obtener la funcionalidad requerida.

Comenzamos con la clase “CProvincia”, se encargará de manejar los datos de las tablas “Provincias” y “Departamentos”.

### Clase “CProvincia”

Esta clase trabajará con las tablas de Provincias y Departamentos y nos permitirá cargar esa información en el control TreeView. Su contenido será este:

```

CargarTreeView(System.Windows.Forms.TreeView)
CProvincia()
Dispose()
ObtenerProvincia(int)
DS
TablaDepartamento
TablaProvincia

```

El código completo de la clase resulta así:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data;
using System.Data.OleDb;
using System.Windows.Forms;

```

```

namespace SP3
{
    public class CProvincia
    {
        DataSet DS;
        String TablaProvincia = "Provincias";
        String TablaDepartamento = "Departamentos";

        // constructor
        public CProvincia()
        {
            try
            {
                OleDbConnection cnn = new OleDbConnection();
                cnn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0; Data
Source=Incendios.mdb";
                cnn.Open();
                DS = new DataSet();
                OleDbCommand cmdP = new OleDbCommand();
                cmdP.Connection = cnn;
                cmdP.CommandType = CommandType.TableDirect;
                cmdP.CommandText = TablaProvincia;
                OleDbDataAdapter DAP = new OleDbDataAdapter(cmdP);
                DAP.Fill(DS, TablaProvincia);
                DataColumn[] pkP = new DataColumn[1];
                pkP[0] = DS.Tables[TablaProvincia].Columns["Provincia"];
                DS.Tables[TablaProvincia].PrimaryKey = pkP;
                //
                OleDbCommand cmdD = new OleDbCommand();
                cmdD.Connection = cnn;
                cmdD.CommandType = CommandType.TableDirect;
                cmdD.CommandText = TablaDepartamento;
                OleDbDataAdapter DAD = new OleDbDataAdapter(cmdD);
                DAD.Fill(DS, TablaDepartamento);
                DataColumn[] pkD = new DataColumn[1];
                pkD[0] = DS.Tables[TablaDepartamento].Columns["Departamento"];
                DS.Tables[TablaDepartamento].PrimaryKey = pkD;
                cnn.Close();
            }
            catch (Exception ex)
            {
                throw new Exception("CProvincia: " + ex.Message);
            }
        }

        public int ObtenerProvincia(int departamento)
        {
            // devuelve el número de provincia de un departamento
            DataRow dr = DS.Tables[TablaDepartamento].Rows.Find(departamento);
            if(dr == null)
            {
                throw new Exception("CProvincia: No existe del departamento");
            }
            return (int)dr["Provincia"];
        }

        public void CargarTreeView(TreeView tvw)
        {
            tvw.Nodes.Clear();
            // agregar la raíz
            TreeNode raiz = tvw.Nodes.Add("raiz", "INCENDIOS", 0, 0);
            try

```

```

    {

        // agregar las provincias
        foreach (DataRow drProv in DS.Tables[TablaProvincia].Rows)
        {
            TreeNode prov = raiz.Nodes.Add(drProv["provincia"].ToString(),
            drProv["Nombre"].ToString(), 1, 1);
            // agregar los departamentos de la provincia
            foreach (DataRow drDep in DS.Tables[TablaDepartamento].Rows)
            {
                // comparar el numero de provincia
                if ((int)drProv["Provincia"] == (int)drDep["Provincia"])
                {
                    // agregar el departamento como nodo hijo de la provincia
                    prov.Nodes.Add(drDep["Departamento"].ToString(),
                    drDep["Nombre"].ToString(), 2, 2);
                }
            }
        }
    }
    catch(Exception ex)
    {
        throw new Exception("CProvincia: " + ex.Message);
    }
}

public void Dispose()
{
    DS.Dispose();
}
}
}

```

En el **constructor** de la clase se realiza la conexión con la base de datos y se cargan en el DataSet los registros de las tablas Provincias y Departamentos, además se asignan las claves primarias de ambas tablas.

El método “**CargarTreeView**” recibe por parámetro el control treeView del formulario, se encarga de agregar el nodo raíz, los nodos con los nombres de las provincias y para cada uno de ellos agrega como hijos los nodos con los nombres de los departamentos.

Observe que en el momento de agregar cada nodo se hace referencia al icono que se usará para el nodo, los valores usados: 0, 1 y 2 son los índices que ocupan los iconos en el control ImageList que contiene los iconos.

```

// agregar la raíz
TreeNode raiz = tvw.Nodes.Add("raiz", "INCENDIOS", 0, 0);

```

```

// agregar una provincia
TreeNode prov =
raiz.Nodes.Add(drProv["provincia"].ToString(), drProv["Nombre"].ToString(), 1, 1);

```

```
// agregar el departamento como nodo hijo de la provincia
prov.Nodes.Add(drDep["Departamento"].ToString(), drDep["Nombre"].ToString(), 2, 2);
```

Para la raíz se usa el índice 0, para las provincias el índice 1 y para los departamentos el índice 2.

El método “**ObtenerProvincia**” recibe por parámetro el número de un departamento y valiéndose de la clave primaria de la tabla Departamentos realiza una búsqueda de ese valor.

```
DataRow dr = DS.Tables[TablaDepartamento].Rows.Find(departamento);
```

Del registro localizado se toma el valor del campo “provincia”, ese número de provincia es el valor que devuelve el método.

```
return (int)dr["Provincia"];
```

Esto nos facilita conocer a qué provincia pertenece cada departamento cuando necesitamos procesar las cantidades de incendios.

El método “**Dispose**” libera los recursos del DataSet usado por la clase.

### Clase “CIncendio”

La clase “CIncendio” trabajará con las tablas de Incendios y TipoIncendio y sus métodos estarán relacionados con los procesos que deben cargar el detalle de cantidad de incendios por tipo de incendio en el control ListView del formulario. Su contenido es este:

```
CIncendio()
Dispose()
ObtenerIncendios(System.Windows.Forms.ListView)
ObtenerIncendiosPorDepartamento(int, System.Windows.Forms.ListView)
ObtenerIncendiosPorProvincia(int, System.Windows.Forms.ListView)
DS
TablaIncendio
TablaTipoIncendio
```

Además de los métodos para el constructor de la clase y Dispose vamos a implementar 3 métodos para cargar el control ListView, los 3 métodos deben mostrar todos los tipos de incendio que existen, y para cada tipo de incendio la cantidad total de incendios registrado, la diferencia entre esos 3 métodos será el nivel de detalle de los totales obtenidos, para el nodo raíz se mostrarán los totales generales, para un nodo de provincia los totales serán con las

cantidades registradas solamente para esa provincia y para un nodo de departamento las cantidades serán las que registre sólo ese departamento.

En todos los casos necesitamos recorrer primero la tabla de TipoDepartamento y por cada tipo luego recorrer la tabla de Incendios.

El contenido completo de la clase “CIncendio” es este:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data;
using System.Data.OleDb;
using System.Windows.Forms;

namespace SP3
{
    public class CIncendio
    {
        DataSet DS;
        String TablaIncendio = "Incendios";
        String TablaTipoIncendio = "TipoIncendio";

        // constructor
        public CIncendio()
        {
            try
            {
                OleDbConnection cnn = new OleDbConnection();
                cnn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0; Data
Source=Incendios.mdb";
                cnn.Open();
                DS = new DataSet();
                OleDbCommand cmdI = new OleDbCommand();
                cmdI.Connection = cnn;
                cmdI.CommandType = CommandType.TableDirect;
                cmdI.CommandText = TablaIncendio;
                OleDbDataAdapter DAI = new OleDbDataAdapter(cmdI);
                DAI.Fill(DS, TablaIncendio);
                DataColumn[] pkI = new DataColumn[2];
                pkI[0] = DS.Tables[TablaIncendio].Columns["Departamento"];
                pkI[1] = DS.Tables[TablaIncendio].Columns["TipoIncendio"];
                DS.Tables[TablaIncendio].PrimaryKey = pkI;
                //
                OleDbCommand cmdTI = new OleDbCommand();
                cmdTI.Connection = cnn;
                cmdTI.CommandType = CommandType.TableDirect;
                cmdTI.CommandText = TablaTipoIncendio;
                OleDbDataAdapter DATI = new OleDbDataAdapter(cmdTI);
                DATI.Fill(DS, TablaTipoIncendio);
                DataColumn[] pkTI = new DataColumn[1];
                pkTI[0] = DS.Tables[TablaTipoIncendio].Columns["TipoIncendio"];
                DS.Tables[TablaTipoIncendio].PrimaryKey = pkTI;
                cnn.Close();
            }
            catch (Exception ex)
            {
            }
        }
    }
}
```

```

        String MsgErr = "CIncendio: " + ex.Message;
        throw new Exception(MsgErr);
    }
}

public int ObtenerIncendiosPorDepartamento(int departamento, ListView lvw)
{
    int Total = 0; // cantidad total de incendios del departamento
    int Cantidad = 0; // cantidad de incendios por tipo

    lvw.Items.Clear();
    // recorrer los tipos de Incendio
    foreach (DataRow dr in DS.Tables[TablaTipoIncendio].Rows)
    {
        Cantidad = 0;
        // texto del item (primera columna de LsitView)
        ListViewItem item = lvw.Items.Add(dr["Descripcion"].ToString());
        // recorrer la tabla de incendios
        foreach (DataRow drI in DS.Tables[TablaIncendio].Rows)
        {
            if( departamento == (int)drI["Departamento"] &&
                (int)drI["TipoIncendio"] == (int)drI["TipoIncendio"])
            {
                Cantidad += (int)drI["Cantidad"];
            }
        }

        // texto del subitem (segunda columna)
        item.SubItems.Add(Cantidad.ToString());
        // acumular el total
        Total += Cantidad;
    }

    return Total;
}

public int ObtenerIncendiosPorProvincia(int provincia, ListView lvw)
{
    int Total = 0; // cantidad total de incendios de la provincia
    int Cantidad = 0; // cantidad de incendios por tipo
    int prov_dep;
    CProvincia prov = new CProvincia();

    lvw.Items.Clear();
    // recorrer los tipos de Incendio
    foreach (DataRow dr in DS.Tables[TablaTipoIncendio].Rows)
    {
        Cantidad = 0;
        // texto del item (primera columna de LsitView)
        ListViewItem item = lvw.Items.Add(dr["Descripcion"].ToString());
        // recorrer la tabla de incendios
        foreach (DataRow drI in DS.Tables[TablaIncendio].Rows)
        {
            prov_dep = prov.ObtenerProvincia((int)drI["Departamento"]);

            if (provincia == prov_dep &&
                (int)drI["TipoIncendio"] == (int)drI["TipoIncendio"])
            {
                Cantidad += (int)drI["Cantidad"];
            }
        }
    }
}

```

```

        // texto del subitem (segunda columna)
        item.SubItems.Add(Cantidad.ToString());
        // acumular el total
        Total += Cantidad;
    }
    prov.Dispose();
    return Total;
}

public int ObtenerIncendios(ListView lvw)
{
    int Total = 0; // cantidad total de incendios
    int Cantidad = 0; // cantidad de incendios por tipo

    lvw.Items.Clear();
    // recorrer los tipos de Incendio
    foreach (DataRow dr in DS.Tables[TablaTipoIncendio].Rows)
    {
        Cantidad = 0;
        // texto del item (primera columna de ListView)
        ListViewItem item = lvw.Items.Add(dr["Descripcion"].ToString());
        // recorrer la tabla de incendios
        foreach (DataRow drI in DS.Tables[TablaIncendio].Rows)
        {
            if ((int)dr["TipoIncendio"] == (int)drI["TipoIncendio"])
            {
                Cantidad += (int)drI["Cantidad"];
            }
        }

        // texto del subitem (segunda columna)
        item.SubItems.Add(Cantidad.ToString());
        // acumular el total
        Total += Cantidad;
    }

    return Total;
}

public void Dispose()
{
    DS.Dispose();
}
}
}

```

En el **constructor** de la clase se realiza la conexión con la base de datos y se cargan en el DataSet los registros de las tablas Incendios y TipoIncendio, además se asignan las claves primarias de ambas tablas.

El método “**ObtenerIncendiosPorDepartamento**” recibe dos parámetros, el primero es el número de departamento consultado y el segundo parámetro es el control ListView que se debe cargar. Como se mencionó anteriormente, el proceso debe recorrer primero la tabla

“TipoIncendio” ya que en el ListView hay que agregar una fila para cada tipo de incendio, luego se deberá recorrer la tabla de Incendios y buscar los registros que sean del tipo adecuado y que además pertenezcan a departamento consultado (hay una doble condición que debe cumplirse):

```
if( departamento == (int)drI["Departamento"] &&
    (int)drI["TipoIncendio"] == (int)drI["TipoIncendio"])
{
    Cantidad += (int)drI["Cantidad"];
}
```

Para cada tipo de incendio se obtiene una cantidad que se muestra en la segunda columna del control ListView, además esas cantidades se acumulan en un total:

```
// acumular el total
Total += Cantidad;
```

El método finaliza devolviendo el total obtenido para que pueda ser mostrado en el control StatusStrip del formulario.

El método “**ObtenerIncendiosPorProvincia**” es similar al anterior en cuanto al proceso de las tablas, pero acá se recibe por parámetro el número de una provincia, entonces los totales de incendios para cada tipo de incendio serán la suma de las cantidades de todos los departamentos que pertenecen a esa provincia.

Como el contenido de la tabla de Incendios está dado por departamento vamos a necesitar consultar a qué provincia pertenece cada departamento y así poder decidir si se suma o no su cantidad. Usaremos entonces un objeto de la clase “CProvincia” y el método “ObtenerProvincia” para resolver ese punto.

```
prov_dep = prov.ObtenerProvincia((int)drI["Departamento"]);

if (provincia == prov_dep &&
    (int)drI["TipoIncendio"] == (int)drI["TipoIncendio"])
{
    Cantidad += (int)drI["Cantidad"];
}
```

Como podemos ver acá también tenemos una doble condición que se debe cumplir, que el registro del incendio sea de la provincia consultada y que además sea del mismo tipo de incendio que se está procesando en ese momento.



El método “**ObtenerIncendios**” se usará para obtener las cantidades totales de incendios sin distinguir por provincia o departamento, por lo que resulta un poco más sencillo.

La condición usada para sumar las cantidades en este caso es simple, el registro sólo debe pertenecer al tipo de incendio que se está procesando:

```
if ((int)dr["TipoIncendio"] == (int)drI["TipoIncendio"])
{
    Cantidad += (int)drI["Cantidad"];
}
```

Finalmente tenemos el método “**Dispose**” que como en la clase anterior se encarga de liberar los recursos usados por el objeto DataSet.

De esta forma queda completada la implementación de las dos clases auxiliares para resolver la situación propuesta, nos resta desarrollar el código del formulario que en este caso será bastante simple ya que prácticamente se limitará a crear los objetos y ejecutar sus métodos.

### “Form1”

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace SP3
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            CProvincia provincia = new CProvincia();
            provincia.CargarTreeView(tvwIncendios);
            provincia.Dispose();
        }

        private void tvwIncendios_AfterSelect(object sender, TreeViewEventArgs e)
        {
            CIncendio incendio = new CIncendio();
            int Total;
            // determinar el nivel del nodo seleccionado
            TreeNode nodo = e.Node;
```

```

switch (nodo.Level)
{
    case 0: // es el nodo raiz
        Total = incendio.ObtenerIncendios(lvwCantidades);
        sspEstado.Items["lblTotal"].Text = Total.ToString();
        break;

    case 1: // es un nodo de Provincia
        Total =
incendio.ObtenerIncendiosPorProvincia(int.Parse(nodo.Name), lvwCantidades);
        sspEstado.Items["lblTotal"].Text = "Provincia " + nodo.Text + ": "
+ Total.ToString();
        break;

    case 2: // es un nodo de Departamento
        Total =
incendio.ObtenerIncendiosPorDepartamento(int.Parse(nodo.Name), lvwCantidades);
        sspEstado.Items["lblTotal"].Text = "Departamento " + nodo.Text +
": " + Total.ToString();
        break;
    }
    // liberar los recursos
    incendio.Dispose();
}
}
}

```

En el evento “**Load**” del formulario vamos a usar un objeto de la clase “CProvincia” para ejecutar el método que cargará el control TreeView con los nodos de las provincias y departamentos.

```

CProvincia provincia = new CProvincia();
provincia.CargarTreeView(tvwIncendios);
provincia.Dispose();

```

En el evento “**AfterSelect**” del TreeView se toma una referencia al nodo que generó el evento, y con una estructura “switch” se evalúa el valor de la propiedad “Level” (nivel), de ahí surgen 3 posibles valores: 0 si es el nodo raíz, 1 si es un nodo de provincia y 2 si es un nodo de departamento.

```

switch (nodo.Level)
{
    case 0: // es el nodo raiz
        Total = incendio.ObtenerIncendios(lvwCantidades);
        sspEstado.Items["lblTotal"].Text = Total.ToString();
        break;

    case 1: // es un nodo de Provincia
        Total =
incendio.ObtenerIncendiosPorProvincia(int.Parse(nodo.Name), lvwCantidades);
        sspEstado.Items["lblTotal"].Text = "Provincia " + nodo.Text + ": " +
Total.ToString();
        break;

    case 2: // es un nodo de Departamento

```

```

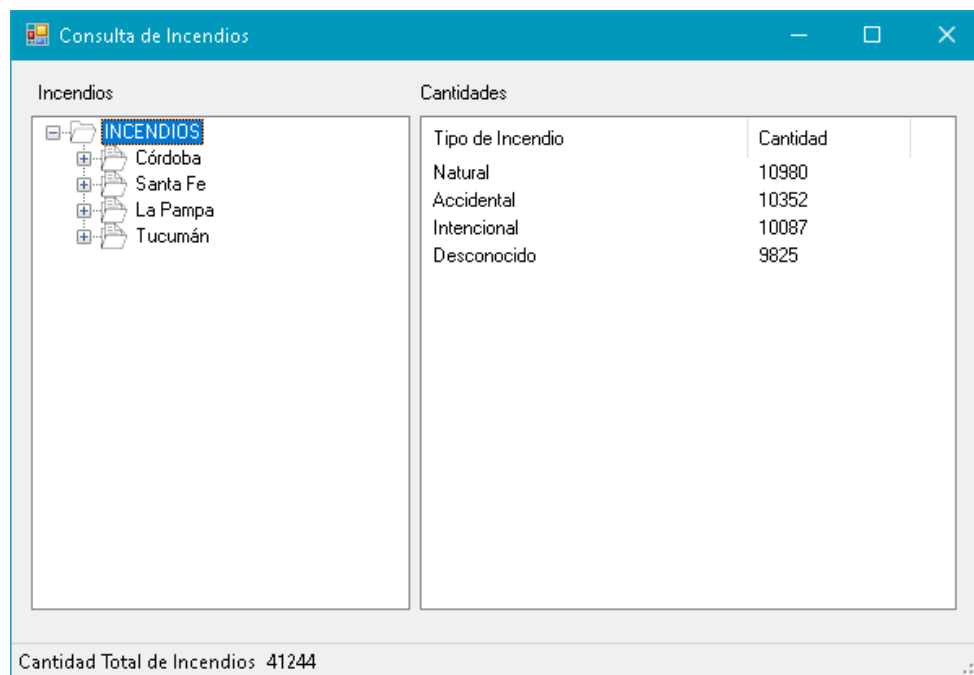
        Total =
        incendio.ObtenerIncendiosPorDepartamento(int.Parse(nodo.Name), lvwCantidades);
        sspEstado.Items["lblTotal"].Text = "Departamento " + nodo.Text + ": " +
Total.ToString();
        break;
    }

```

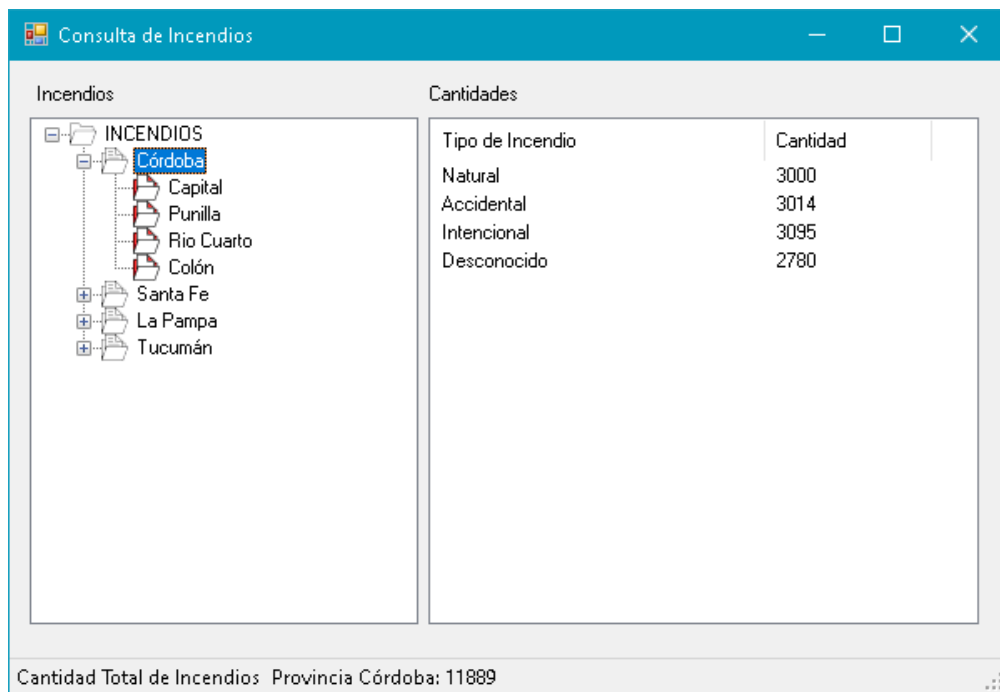
De esa forma se determina qué método de la clase “CIncendio” será invocado para cargar el control ListView con el detalle de las cantidades correspondiente a cada caso.

El valor del total de incendios que devuelven esos métodos se asigna al label del control StatusStrip junto con el nombre de la provincia o departamento según el caso.

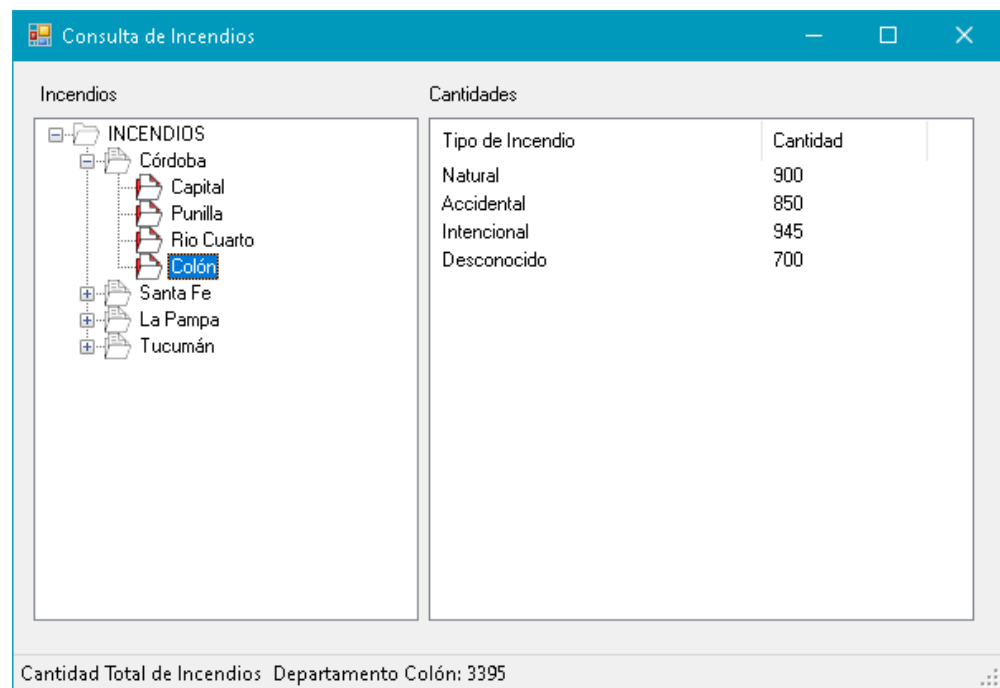
A continuación, vemos el resultado de ejecutar la aplicación y seleccionar el nodo raíz:



Si seleccionamos el nodo de una provincia:



Y si seleccionamos el nodo de un departamento:



Como se puede verificar, en cada caso el ListView y el StatusStrip se actualizan de acuerdo a lo solicitado.

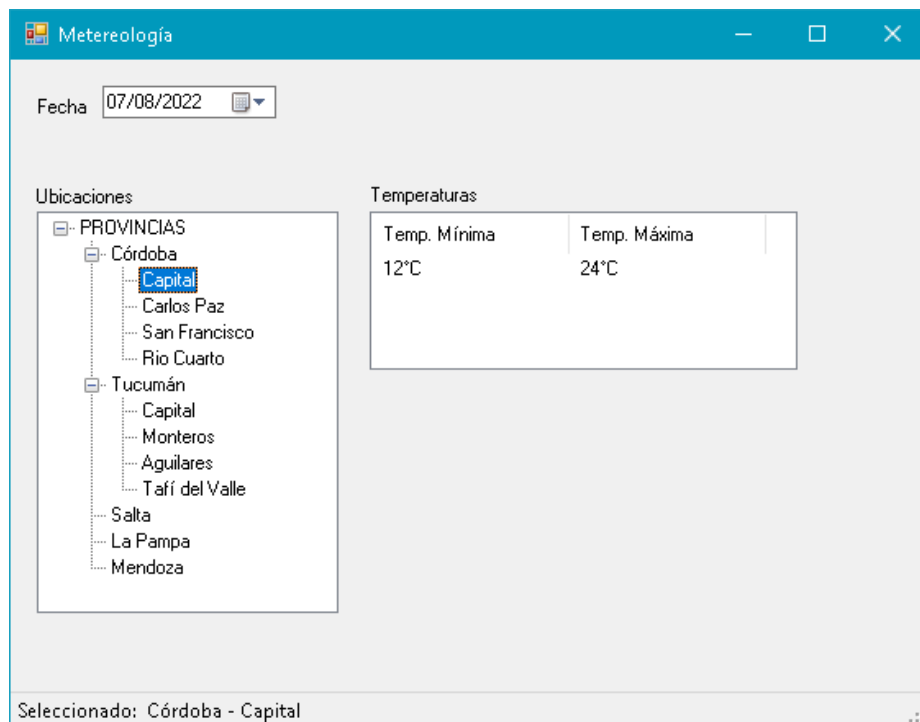
## SP3/Ejercicio por resolver

El administrador de una agencia dependiente del Servicio de Meteorología, le ha solicitado a usted que, en calidad de pasante de la agencia, desarrolle una aplicación para consultar las temperaturas mínimas y máximas registradas en las localidades de las provincias de Argentina en todos los días del año.

Esta información será utilizada para conocer la temperatura mínima y máxima en una fecha dada de una localidad determinada o de todas las localidades de una provincia en particular.

Para la consulta de la información, primero seleccionar una fecha determinada de un control “DateTimePicker” luego, en un control de tipo TreeView se podrá seleccionar un nodo con el nombre de una provincia o un nodo con el nombre de una localidad. El control TreeView estará cargado inicialmente con todas las provincias y para cada provincia tendrá todas las localidades que le pertenecen.

Cada vez que el usuario seleccione un nodo en el TreeView la aplicación deberá mostrar en un control ListView las temperaturas mínimas y máximas de la provincia o de la localidad en la fecha seleccionada. Las columnas del ListView son: la temperatura mínima y la temperatura máxima.



En un control StatusStrip se mostrarán los nombres de la provincia y localidad seleccionados.

La base de datos ya contiene almacenada los datos de algunas las provincias y sus principales localidades de Argentina como así también las temperaturas en cada localidad en distintas fechas.

La tabla provincias almacena un número para identificar a la provincia y el nombre de la provincia. La clave principal es el número de provincia.

La tabla localidades almacena un número para identificar a la localidad, el nombre de la localidad y el número de provincia a la que pertenece la localidad. La clave principal es el número de localidad.

La tabla temperaturas almacena un número para identificar a la localidad, la fecha, la temperatura mínima y la temperatura máxima. La clave principal está compuesta por las columnas número de localidad y fecha.

**1. Indique la opción correcta**

Para cambiar la vista, en un ListView, se puede usar la propiedad View.

Verdadero     X

Falso

**2. Indique la opción correcta**

¿Cuál es el objeto que nos permite ver una barra de progreso?

ToolStripProgressBar     X

ToolStripDropDownButton

ToolStripProgress

ToolStripButton

**3. Indique la opción correcta**

La propiedad Nodes del control TreeView contiene la lista de nodos del nivel superior de la vista de árbol.

Verdadero     X

Falso

**4. Indique la opción correcta**

Para mostrar íconos en tamaño grande dentro de un ListView debemos utilizar la propiedad:

BigIcon

LargeIcon     X

IconLarge

Details

**5. Indique la opción correcta**

Un treeview sirve para crear un árbol que muestre, al menos, dos niveles de una base de datos.

Verdadero      X

Falso

#### **6. Indique la opción correcta**

El metodo LastNode obtiene el último nodo secundario en la colección de nodos que se almacena en la propiedad Nodes del actual nodo.

Verdadero      X

Falso

#### **7. Indique la opción correcta**

Un treeview puede contener muchos nodos raíz.

Verdadero      X

Falso

#### **8. Indique la opción correcta**

En un ToolStrip la propiedad DisplayStyle muestra el texto y las imágenes en un ToolStripButton.

Verdadero

Falso      X