Utilizando la clase MemoryStream con imágenes en GTK# y MonoDevelop

por Martín A. O Márquez <xomalli@gmail.com>

El mecanismo de comunicación entre las aplicaciones .NET y los dispositivos de entrada y de salida se realiza mediante streams (flujos secuenciales) de bytes que se obtienen de una fuente de entrada hacia la aplicación y salen de la aplicación hacia una fuente de salida, pensemos en los flujos como corrientes de agua que van de un recipiente a otro.

Dentro del ensamblado System.IO existe la clase abstracta Stream que implementa operaciones de lectura y escritura sincrónica y asíncrona y que sirve como base para cualquier clase derivada que sea utilizada en operaciones de entrada y salida, como ejemplos de clases derivadas tenemos a FileStream, CriptoStream, NetworkStream, BufferedStream y MemoryStream. De estas clases la clase MemoryStream proporciona una área de almacenamiento temporal en memoria o sea un arreglo de bytes sin signo, entre sus usos más comunes se encuentran las operaciones con imágenes, la compresión y descompresión de archivos, el cifrado/descifrado de datos y la serialización entre otros.

Algunas propiedades y métodos de la clase MemoryStream:

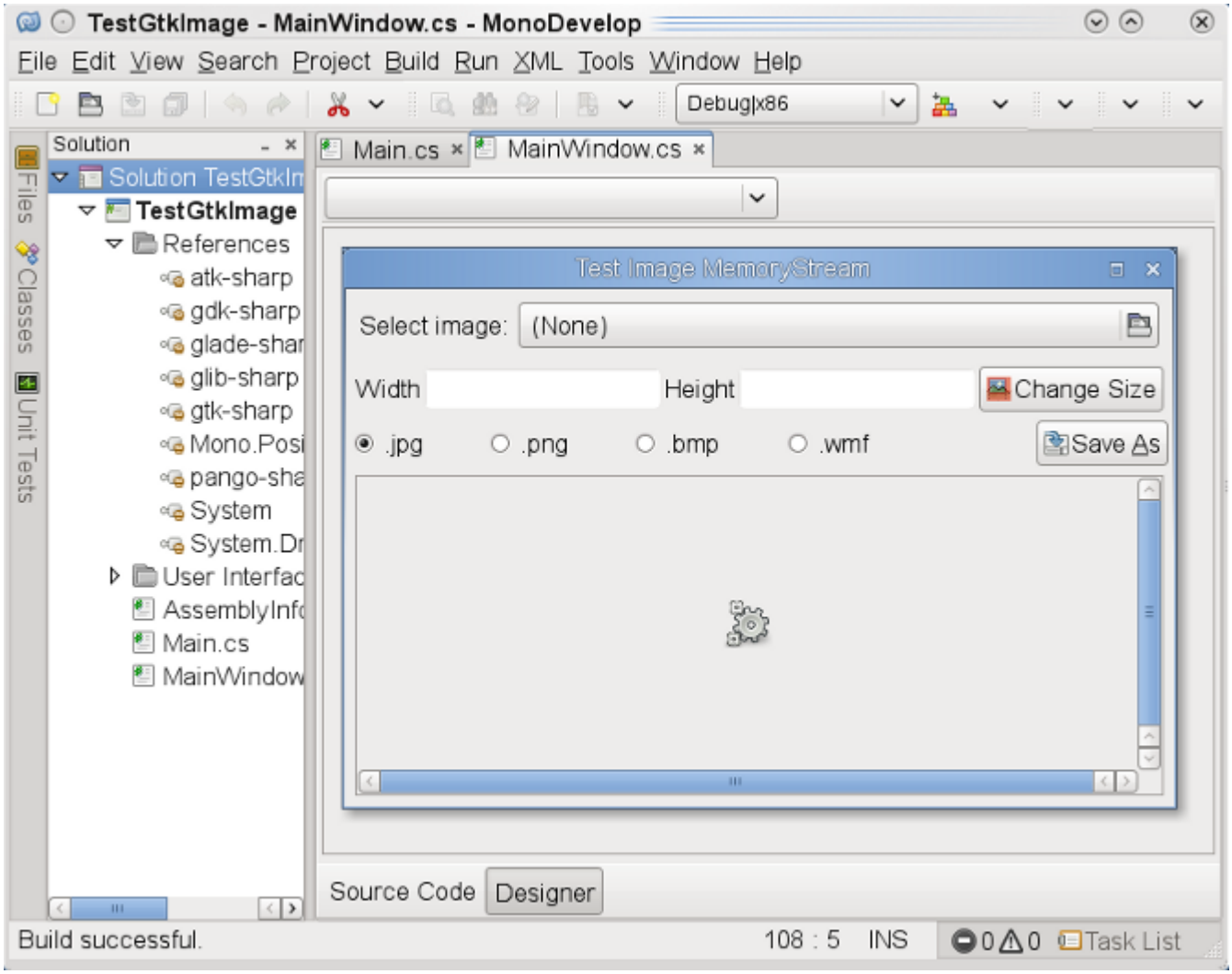
Read: Inicia la operación de lectura de forma secuencial de un número de bytes desde el inicio hasta el final del flujo.

- **ReadByte:** Idéntico que Read solo que lee un byte de forma secuencial.
- **Length:** Es la longitud en bytes del flujo.
- **Position:** Obtiene la posición del apuntador lector del flujo.
- **ReadTimeOut:** Se refiere al tiempo de espera para operaciones de lectura.
- **WriteTimeOut:** Igual que ReadTimeOut solo que para operaciones de escritura.
- **Flush:** Vacia el contenido de los buffers del flujo y obliga a que se escriban en el flujo.
- **GetBuffer:** Regresa un arreglo de bytes sin signo que fueron utilizados para crear el flujo.
- **Seek:** Coloca el apuntador lector del flujo en una determinada posición.

La clase MemoryStream se considera una buena solución siempre que se trate con grandes cantidades de datos y se requiera una rápida lectura y escritura de los mismos. Un detalle importante es que si en el constructor de la clase se establece el tamaño del arreglo, después no es posible cambiar el tamaño. Por eso es mejor utilizar un constructor vacío e ir utilizando el arreglo según se requiera.

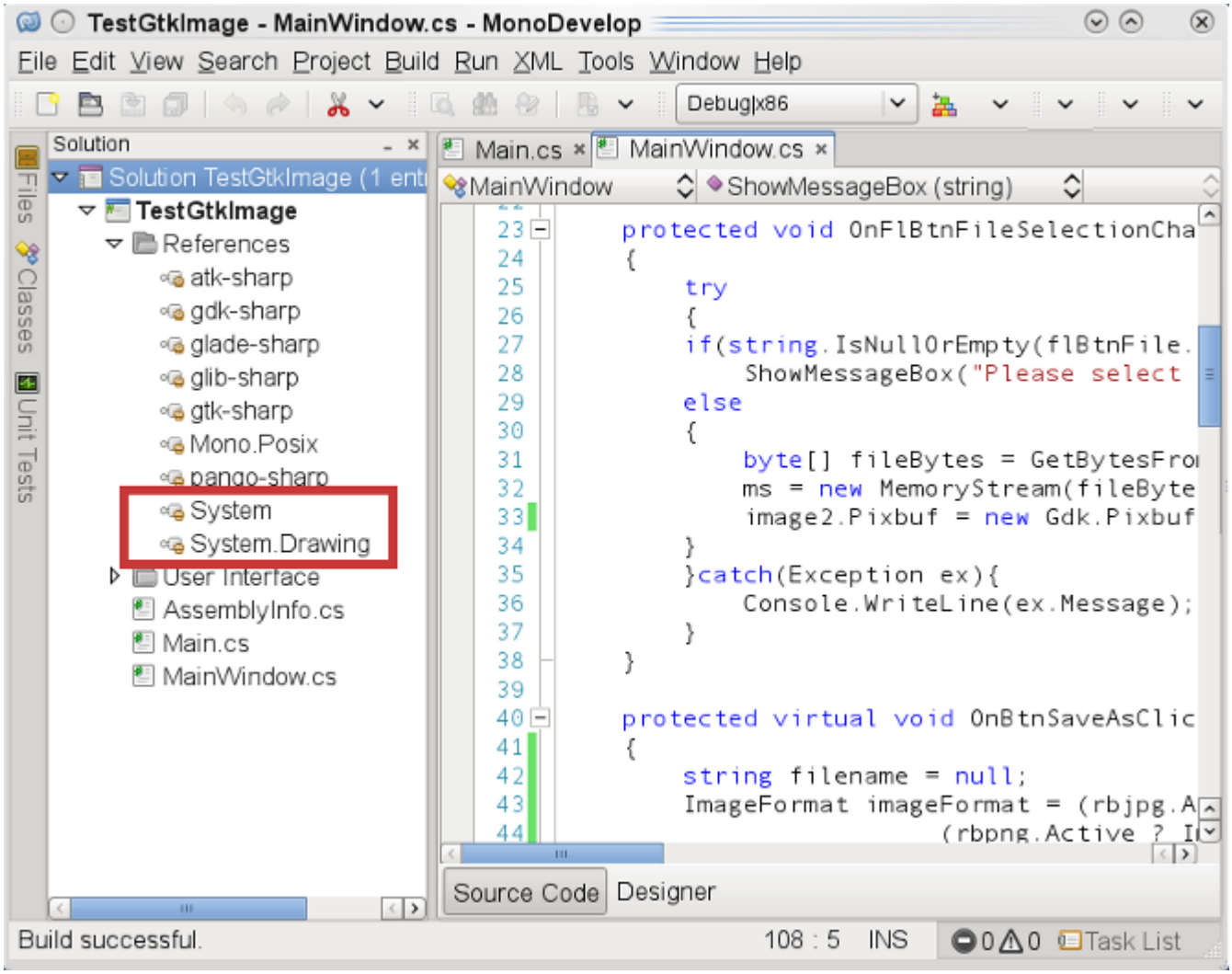
Como ejemplo de su uso, utilizamos una sencilla proyecto GTK# de monodevelop que lee una imagen del disco duro, crea una copia de la imagen en un flujo memorystream para que esa imagen pueda ser ajustada a las medidas que el usuario proporcione y después pueda guardarla con el formato que elija de la aplicación. El diseño del GUI (Graphical User Interface) de la aplicación se muestra en la siguiente imagen, utilizando el designer de MonoDevelop.

Fig 1 Diseñando la interfaz gráfica (GUI) con Monodevelop



Para este proyecto es necesario hacer referencias a los ensamblados: *System.Drawing*, *System.Drawing.Image*, *System.IO* como se muestra en la siguiente imagen:

Fig 2 Agregando las referencias a la solución



El listado completo de la clase principal es el siguiente:

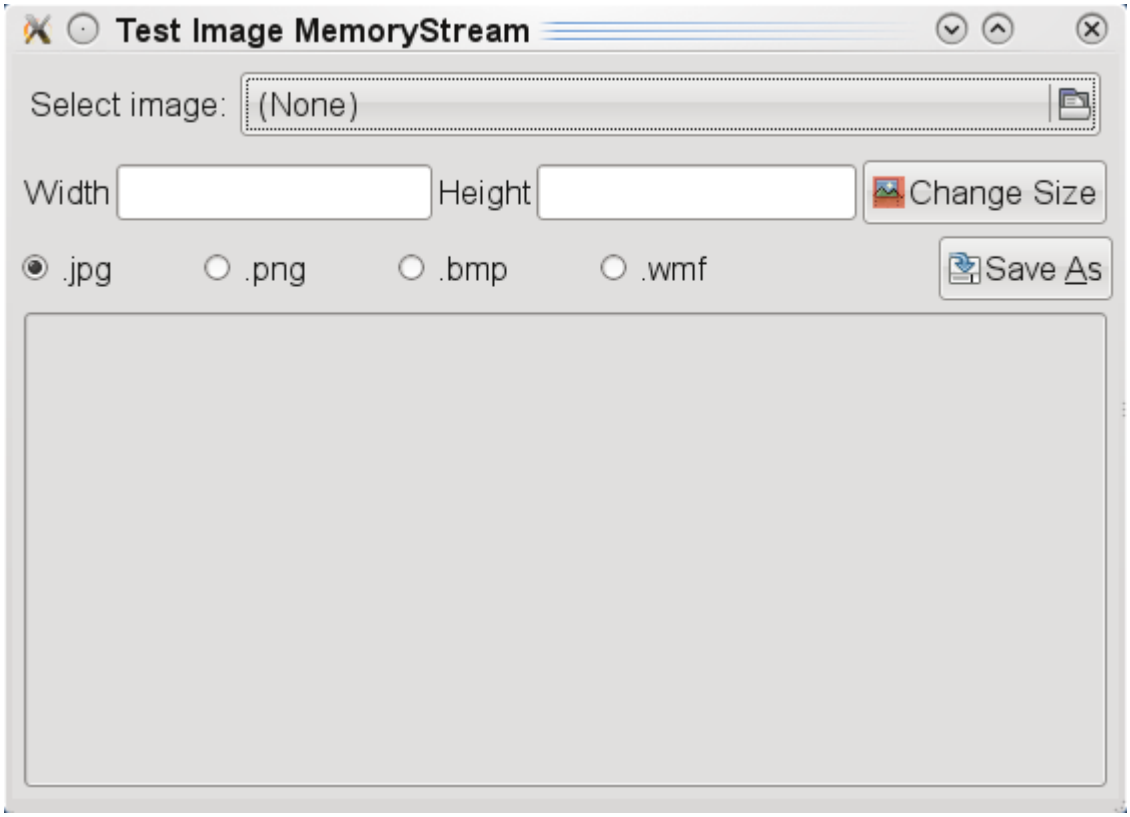
```
 using System;
using Gtk;
using System.IO;
using System.Drawing;
using System.Drawing.Imaging;


public partial class MainWindow: Gtk.Window
{
 MemoryStream ms = null;
 public MainWindow (): base (Gtk.WindowType.Toplevel)
 {
  Build ();
 }

 protected void OnDeleteEvent (object sender, DeleteEventArgs a)
 {
  Application.Quit ();
  a.RetVal = true;
 }
```

```
 protected void OnFlBtnFileSelectionChanged (object sender, EventArgs e)
 {
  try
  {
  if(string.IsNullOrEmpty(flBtnFile.Filename))
   ShowMessageBox("Please select a file.");
  else
  {
   byte[] fileBytes = GetBytesFromFile(flBtnFile.Filename);
   ms = new MemoryStream(fileBytes);
   image2.Pixbuf = new Gdk.Pixbuf(ms.ToArray());
  }
  }catch(Exception ex){
   Console.WriteLine(ex.Message);
  }
 }

 protected virtual void OnBtnSaveAsClicked (object sender, System.EventArgs e)
 {
  string filename = null;
  ImageFormat imageFormat = (rbjpg.Active ? ImageFormat.Jpeg : (rbwmf.Active ?
ImageFormat.Wmf:
                (rbpng.Active ? ImageFormat.Png : ImageFormat.Bmp)));
  try{
   Gtk.FileChooserDialog fc=  new Gtk.FileChooserDialog("Save image",
                              this,
                              FileChooserAction.Save,
                              "Cancel",ResponseType.Cancel,
                              "Save",ResponseType.Accept);
   if (fc.Run() == (int)ResponseType.Accept) {
    filename = fc.Filename + "." + imageFormat.ToString();
    fc.Hide();
    fc.Dispose();
    using(FileStream foutput =
new FileStream(filename,FileMode.Create,FileAccess.Write))
    {
     Bitmap bmp = new Bitmap(ms);
     bmp.Save(foutput,imageFormat);
    }
   }
   else{
    fc.Hide();
    fc.Dispose();
   }

  }catch(Exception ex){
   ShowMessageBox(ex.Message);
  }
 }


 protected virtual void OnBtnMakeThumbClicked (object sender, System.EventArgs
e)
 {
  int w,h;
  try{
   if(!string.IsNullOrEmpty(txtWidth.Text) &&
      !string.IsNullOrEmpty(txtHeight.Text))
   {
    w = Convert.ToInt32(txtWidth.Text);
    h = Convert.ToInt32(txtHeight.Text);
    Bitmap bitmap = new Bitmap(ms);
    System.Drawing.Image.GetThumbnailImageAbort thumbnailCallback =
```

```
    new System.Drawing.Image.GetThumbnailImageAbort(delegate(){ return
false; });
    System.Drawing.Image thumbnail =
bitmap.GetThumbnailImage(w,h,thumbnailCallback,IntPtr.Zero);
    byte[] thumbnailBytes =
((byte[])new ImageConverter().ConvertTo(thumbnail,typeof(byte[])));
    ms = new MemoryStream(thumbnailBytes);
    image2.Pixbuf = new Gdk.Pixbuf(ms.ToArray());
   }
   else
    ShowMessageBox("width & height must be integers.");
  }catch(ApplicationException ex){
   ShowMessageBox(ex.Message);
  }
 }

 byte[] GetBytesFromFile(string filename){
  byte[] imgBytes;
  using(FileStream fis = new FileStream(filename,FileMode.Open,FileAccess.Read))
  {
   BinaryReader reader = new BinaryReader(fis);
   imgBytes = reader.ReadBytes((int)fis.Length);
   reader.Close();
  }
  return imgBytes;
 }


void ShowMessageBox(string msg){
using (Dialog dialog = new MessageDialog (this,
      DialogFlags.Modal | DialogFlags.DestroyWithParent,
      MessageType.Info,
      ButtonsType.Ok,
      msg)) {
dialog.Run ();
dialog.Hide ();
}
}

}
```

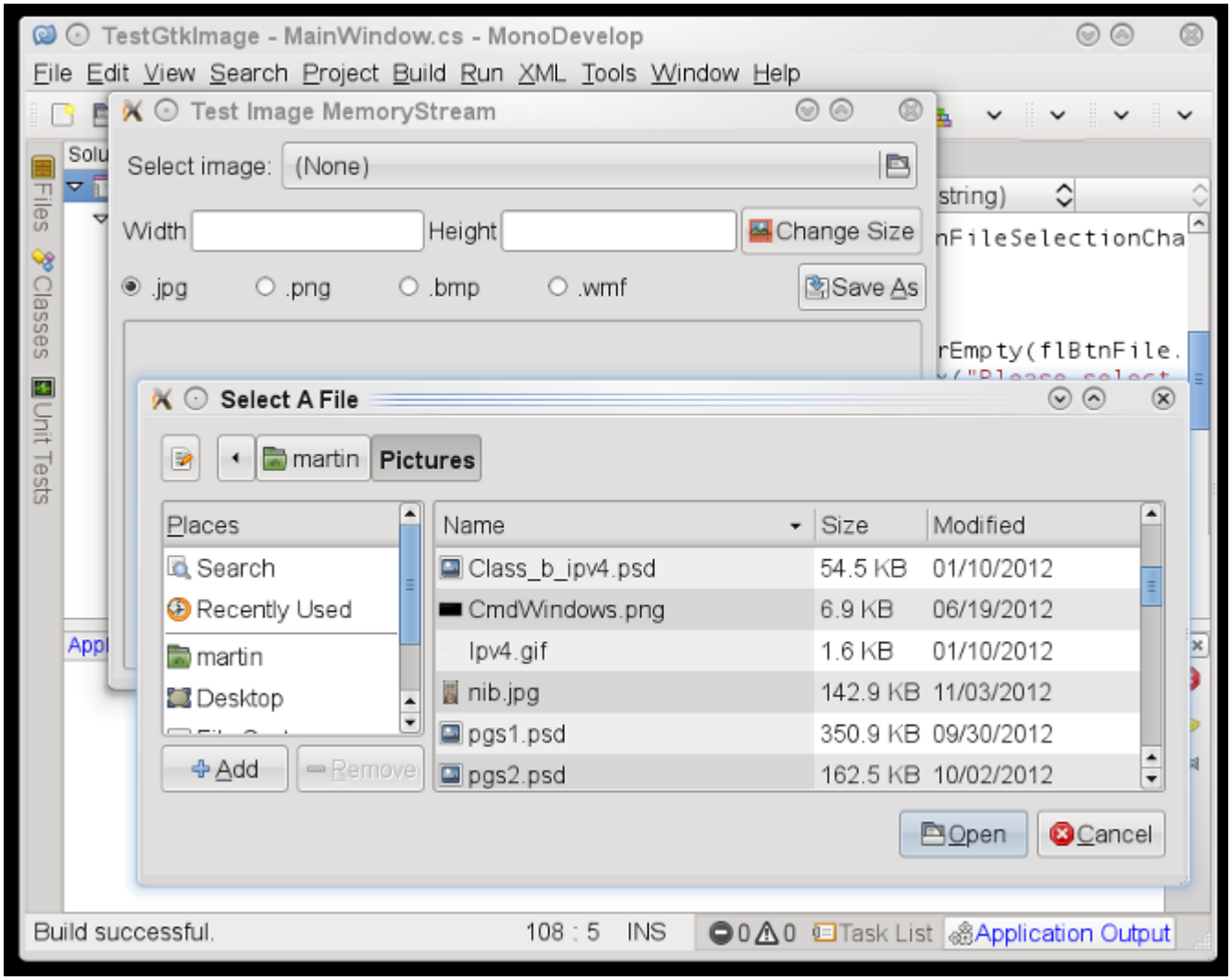
Al oprimir F5 (Debug) ó Ctrl + F5 en Monodevelop se ejecutará la aplicación como se ve en la siguiente imagen:

Fig 3 Ejecutando la aplicación



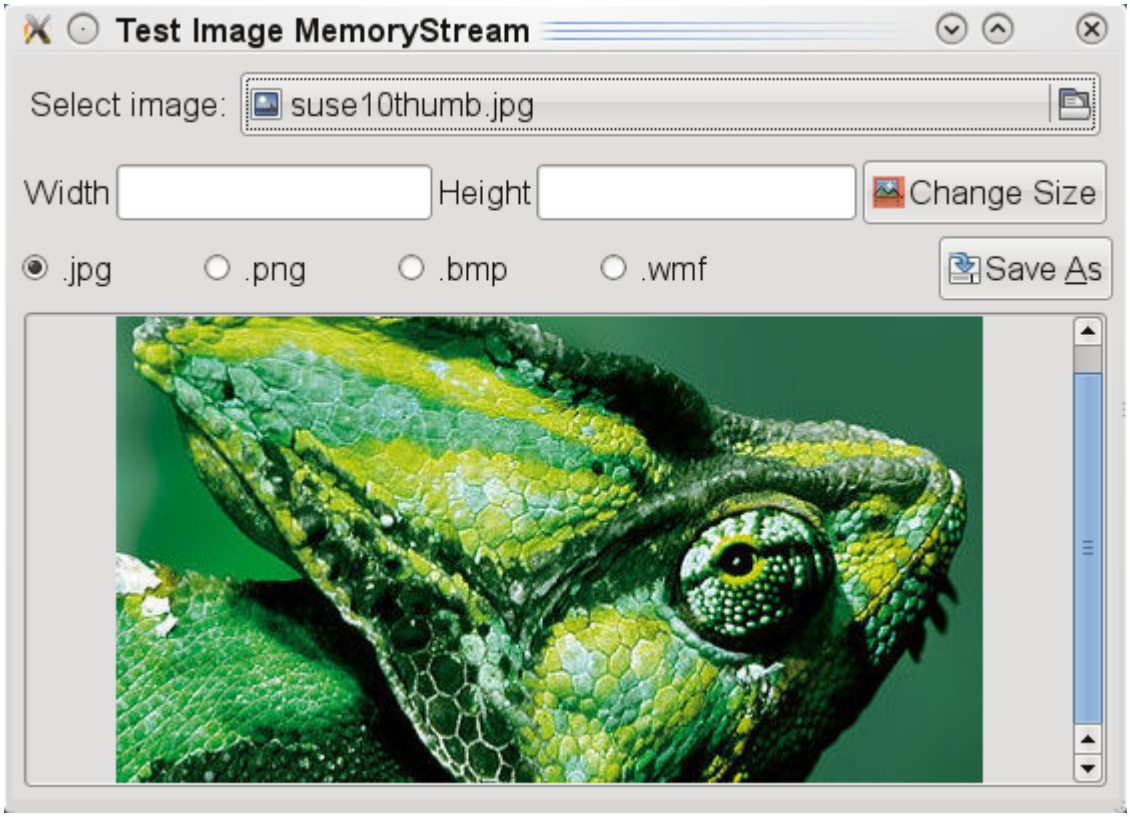
Cuando se presiona el botón de “Select image” se ve el FileDialog para escoger la imagen desde el disco duro:

Fig 4 Seleccionando la imagen



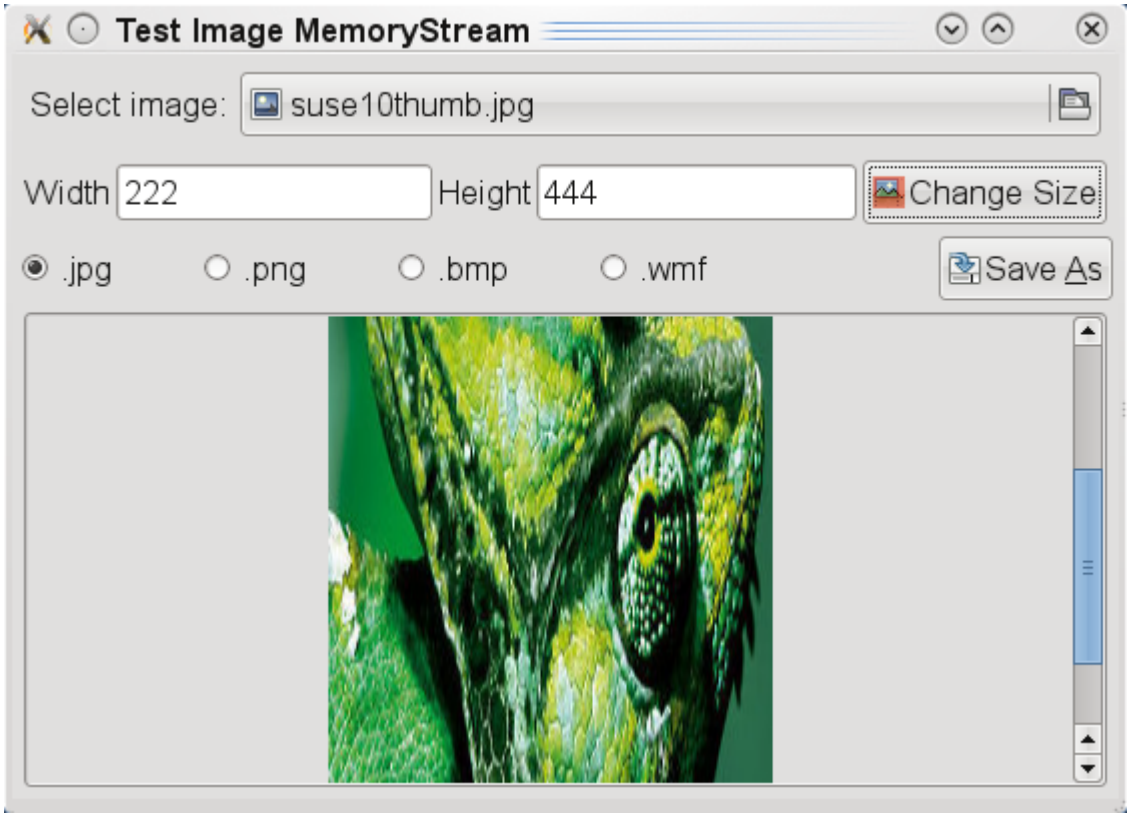
Una vez seleccionada la imagen se mostrará dentro del programa, aunque esa imagen no son los bytes originales del archivo sino los bytes que están contenidos en el buffer del objeto MemoryStream.

Fig 5 Mostrando la imagen en el buffer



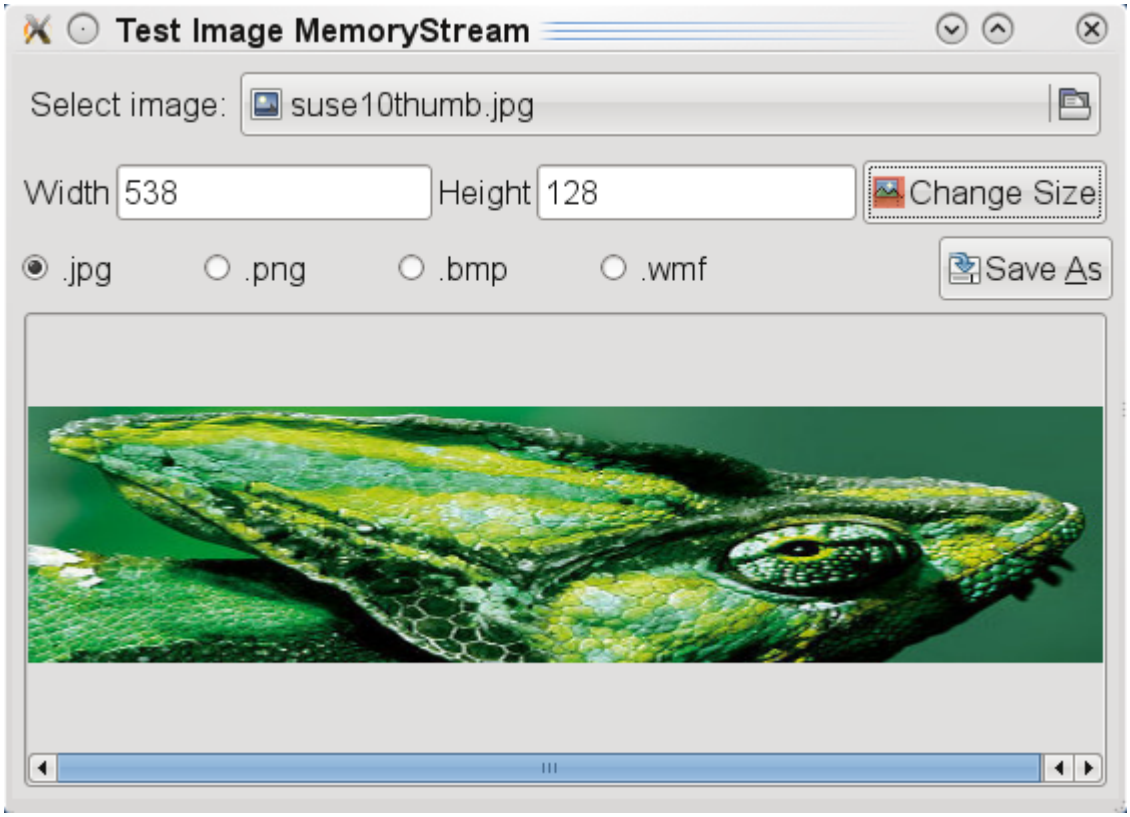
Una vez cargada la imagen puedes redimensionarla (en pixeles) de acuerdo a la anchura y la altura que teclees en los valores de los campos de texto width y height. Al presionar el botón “Change size”la imagen se ajustará a las nuevas dimensiones.

Fig 6 Redimensionando la imagen



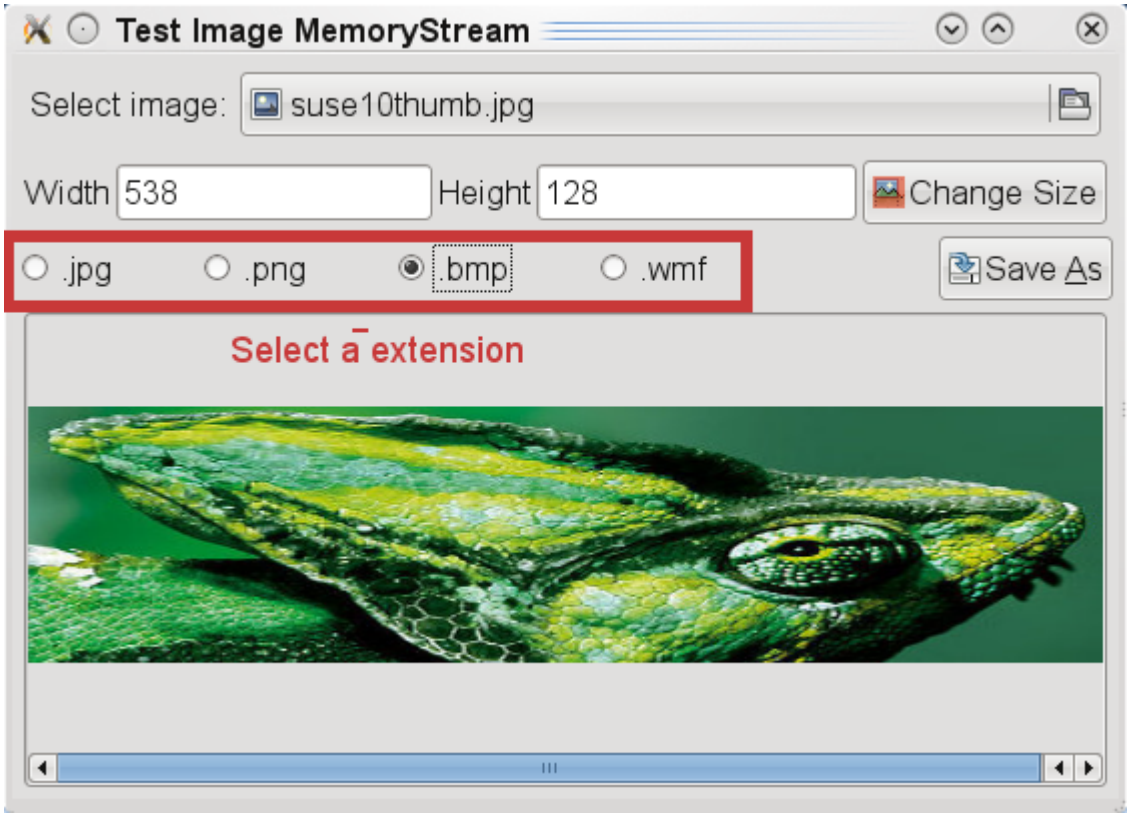
Una segunda imagen como ejemplo con diferentes medidas:

Fig 7 Una imagen redimensionada



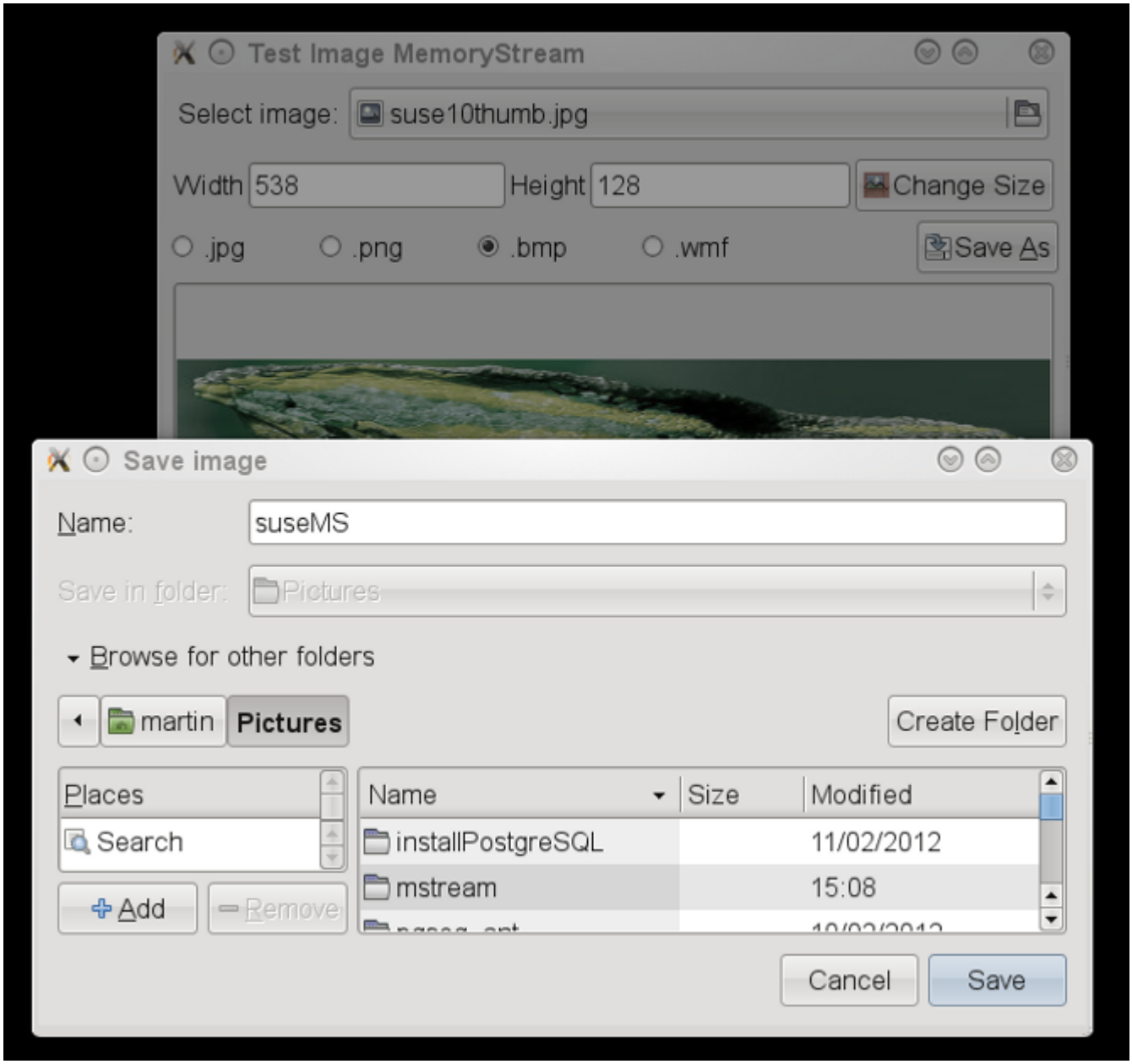
Puedes guardar la imagen modificada con utilizando uno de los formatos que están en los *radiobuttons*

Fig 8 Seleccionar el formato para guardar la imagen



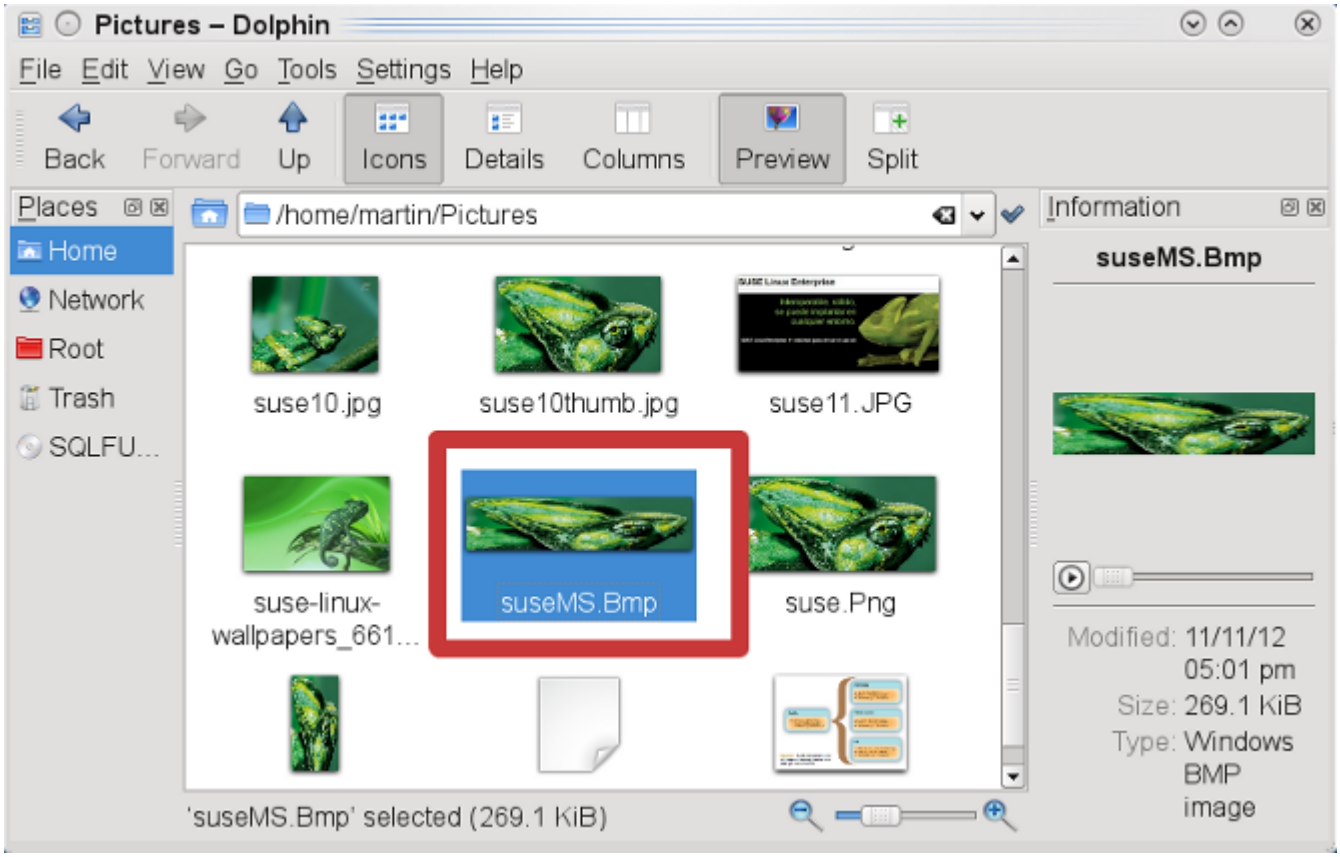
Al presionar el botón *"Save As"* se abrirá un cuadro de dialogo (filechooser) para guardar la imagen en cualquier parte del sistema de archivos.

Fig 9 Guardando la imagen en el sistema de archivos.



Podemos ver la imagen guardada en el sistema de archivos.

Fig 10 La imagen en el sistema de archivo



La aplicación inicia cuando se escoge un archivo de imagen del sistema de archivos para obtener su matriz de bytes. Esto lo hacemos con el siguiente código:

```
 byte[] GetBytesFromFile(string filename){
 byte[] imgBytes;
 using(FileStream fis =
new FileStream(filename,FileMode.Open,FileAccess.Read))
 {
  BinaryReader reader = new BinaryReader(fis);
  imgBytes = reader.ReadBytes((int)fis.Length);
  reader.Close();
 }
  return imgBytes;
 }
```

Entonces podemos esa matriz de bytes en el flujo MemoryStream y colocamos su buffer en el control (widget) Image.

```
 ms = new MemoryStream(fileBytes);
image2.Pixbuf = new Gdk.Pixbuf(ms.ToArray());
```

Aquí ya la aplicación esta lista para redimensionar la imagen esto se logra con el código del evento para el botón “Change Size”.

```
 protected virtual void OnBtnMakeThumbClicked (object sender, System.EventArgs
e)
 {
  int w,h;
  try{
   if(!string.IsNullOrEmpty(txtWidth.Text) &&
```

```
      !string.IsNullOrEmpty(txtHeight.Text))
   {
    w = Convert.ToInt32(txtWidth.Text);
    h = Convert.ToInt32(txtHeight.Text);
    Bitmap bitmap = new Bitmap(ms);
    System.Drawing.Image.GetThumbnailImageAbort thumbnailCallback =
    new System.Drawing.Image.GetThumbnailImageAbort(delegate(){ return
false; });
    System.Drawing.Image thumbnail =
bitmap.GetThumbnailImage(w,h,thumbnailCallback,IntPtr.Zero);
    byte[] thumbnailBytes =
((byte[])new ImageConverter().ConvertTo(thumbnail,typeof(byte[])));
    ms = new MemoryStream(thumbnailBytes);
    image2.Pixbuf = new Gdk.Pixbuf(ms.ToArray());
   }
   else
    ShowMessageBox("width & height must be integers.");
  }catch(ApplicationException ex){
   ShowMessageBox(ex.Message);
  }
 }
```

En este código se utiliza el método GetThumbnailImage de la clase Image:

```
 System.Drawing.Image thumbnail =
bitmap.GetThumbnailImage(w,h,thumbnailCallback,IntPtr.Zero);
```

NOTA: para este código hay que poner toda la ruta de los ensamblados para la clase image ya que exiten dos clases Image una en el ensamblado Gtk y otra en el ensambaldo System.Drawing por lo que hay que diferenciarlas o el compilador se confundirá.

De nueva cuenta se convierte la imagen en una matriz de bytes y la ponemos dentro del buffer del objeto MemoryStream, para después ponerla en el widget de imagen, esto lo realiza las siguientes líneas de código:

```
byte[] thumbnailBytes =
((byte[])new ImageConverter().ConvertTo(thumbnail,typeof(byte[])));
 ms = new MemoryStream(thumbnailBytes);
image2.Pixbuf = new Gdk.Pixbuf(ms.ToArray());
```

Por último el código del botón para guardar la imagen en el sistema de archivos con el formato seleccionado en los botones de radio.

```
 protected virtual void OnBtnSaveAsClicked (object sender, System.EventArgs e)
 {
  string filename = null;
  ImageFormat imageFormat = (rbjpg.Active ? ImageFormat.Jpeg : (rbwmf.Active ?
ImageFormat.Wmf:
                (rbpng.Active ? ImageFormat.Png : ImageFormat.Bmp)));
  try{
   Gtk.FileChooserDialog fc=  new Gtk.FileChooserDialog("Save image",
                              this,
                              FileChooserAction.Save,
                              "Cancel",ResponseType.Cancel,
                              "Save",ResponseType.Accept);
   if (fc.Run() == (int)ResponseType.Accept)
{
    filename = fc.Filename + "." + imageFormat.ToString();
    fc.Hide();
    fc.Dispose();
    using(FileStream foutput =
new FileStream(filename,FileMode.Create,FileAccess.Write))
```

```
   {
     Bitmap bmp = new Bitmap(ms);
     bmp.Save(foutput,imageFormat);
    }
   }
   else{
    fc.Hide();
    fc.Dispose();
   }

  }catch(Exception ex){
   ShowMessageBox(ex.Message);
  }
 }
```

Aquí se utiliza la clase FileStream la cual representa un flujo hacia un archivo para guardar la imagen que se construye a partir de la matriz de imagen del buffer del objeto MemoryStream.

```
using(FileStream foutput =
new FileStream(filename,FileMode.Create,FileAccess.Write))
{
 Bitmap bmp = new Bitmap(ms);
 bmp.Save(foutput,imageFormat);
}
```

 **Descarga el código fuente en un proyecto para MonoDevelop o Visual Studio**