First off, in all the documents you change, you'll need to add the `System.IO` namespace, since the `MemoryStream` object is part of that namespace and it will be used throughout:

```
using System.IO;
```

There are only a few files that are required to be changed, however, there are numerous places where you can overload the `AddImage` method to utilize a `MemoryStream` object. I'll detail each required file here, and then give a few examples of the other places you can change.

For ease of reading **ALL** code snippets that need to be added are in color, and existing code is grayed out.

XImage.cs

The first file that needs to be changed is the **`XImage.cs`** file:
`PDFsharp\code\PdfSharp\PdfSharp.Drawing\XImage.cs`

XImage.cs - Change 1: Add a static `FromStream` method for creating an XImage from a stream.

```
...
/// <summary>
/// Creates an image from the specified file.
/// </summary>
/// <param name="path">The path to a BMP, PNG, GIF, JPEG, TIFF, or PDF file.</param>
public static XImage FromFile(string path)
{
      if (PdfReader.TestPdfFile(path) > 0)
            return new XPdfForm(path);
      return new XImage(path);
}

/// <summary>
/// Creates an image from the specified stream.
/// </summary>
/// <param name="stream">A MemoryStream object containing the image data.</param>
/// <returns></returns>
public static XImage FromStream(MemoryStream stream)
{
      return new XImage(stream);
}

/// <summary>
/// Tests if a file exist. Supports PDF files with page number suffix.
/// </summary>
/// <param name="path">The path to a BMP, PNG, GIF, JPEG, TIFF, or PDF file.</param>
public static bool ExistsFile(string path)
{
...
```

Image.cs

The second file that needs to be changed is the **Image.cs** file:
MigraDoc\code\MigraDoc.DocumentObjectModel\MigraDoc.DocumentObjectModel.Shapes\Image.cs

Image.cs - Change 1: Overload the constructor

```
...
/// <summary>
/// Initializes a new instance of the Image class from the specified (file) name.
/// </summary>
public Image(string name)
      : this()
{
      Name = name;
}

/// <summary>
/// Initializes a new instance of the Image class from the supplied MemoryStream
/// </summary>
public Image(MemoryStream imageStream)
      : this()
{
      ImageStream = imageStream;
      Name = String.Empty;
}

#region Methods
/// <summary>
/// Creates a deep copy of this object.
/// </summary>
public new Image Clone()
{
      return (Image)DeepCopy();
}
...
```

Image.cs - Change 2: Add a property to hold the MemoryStream object

```
...
#region Properties
/// <summary>
/// Gets or sets the value for an image in memory.
/// </summary>
public MemoryStream ImageStream
{
      get { return this.imageStream; }
      set { this.imageStream = value; }
}
internal MemoryStream imageStream = null;

/// <summary>
/// Gets or sets the name of the image.
/// </summary>
public string Name
{
...
```

Image.cs - Change 3: Add a utility property to flag if the file is based on a stream or not

```
...
/// Gets or sets a user defined resolution for the image in dots per inch.
/// </summary>
public double Resolution
{
      get { return this.resolution.Value; }
      set { this.resolution.Value = value; }
}
[DV]
internal NDouble resolution = NDouble.NullValue;

/// <summary>
/// Gets or sets a flag to identify that this image is stream based.
/// </summary>
public bool StreamBased
{
      get { return this.streamBased.Value; }
      set { this.streamBased.Value = value; }
}
[DV]
internal NBool streamBased = NBool.NullValue;

#endregion
...
```

Image.cs - Change 4: Modify the `GetFilePath` method to check if the file is stream based or not

```
...
/// <summary>
/// Gets the concrete image path, taking into account the DOM document's DdlFile and
/// ImagePath properties as well as the given working directory (which can be null).
/// </summary>
public string GetFilePath(string workingDir)
{
      // no need to return a file path if the image is stream based
      if (this.StreamBased)
            return String.Empty;

      string filePath = "";

      try
      {
            if (!String.IsNullOrEmpty(workingDir))
                  filePath = workingDir;
...
```

ImageRenderer.cs

The third file that needs to be changed is the **`ImageRenderer.cs`** file:
`MigraDoc\code\MigraDoc.Rendering\MigraDoc.Rendering\ImageRenderer.cs`

ImageRenderer.cs - Change 1: Modify the overridden `Format` method to skip checks on whether the file exists on disk

```
...
internal override void Format(Area area, FormatInfo previousFormatInfo)
{
	if (!image.StreamBased)
	{
		this.imageFilePath = image.GetFilePath(this.documentRenderer.WorkingDirectory);

		if (!XImage.ExistsFile(this.imageFilePath))
		{
			this.failure = ImageFailure.FileNotFound;
			Trace.WriteLine(Messages.ImageNotFound(this.image.Name), "warning");
		}
	}
	ImageFormatInfo formatInfo = (ImageFormatInfo)this.renderInfo.FormatInfo;
	formatInfo.failure = this.failure;
...
```

ImageRenderer.cs - Change 2: Modify the overridden `Render` method to create the XImage from a stream

```
...
internal override void Render()
{
	RenderFilling();

	ImageFormatInfo formatInfo = (ImageFormatInfo)this.renderInfo.FormatInfo;
	Area contentArea = this.renderInfo.LayoutInfo.ContentArea;
	XRect destRect = new XRect(contentArea.X, contentArea.Y, formatInfo.Width, formatInfo

	if (formatInfo.failure == ImageFailure.None)
	{
		Image xImage = null;
		try
		{
			XRect srcRect = new XRect(formatInfo.CropX, formatInfo.CropY, formatInfo.

			if (image.StreamBased)
				xImage = XImage.FromStream(image.ImageStream);
			else
				xImage = XImage.FromFile(formatInfo.ImagePath);

			this.gfx.DrawImage(xImage, destRect, srcRect, XGraphicsUnit.Point); //Pix
...
```

ImageRenderer.cs - Change 3: Modify the `CalculateImageDimensions` method to create the XImage from a stream

```
private void CalculateImageDimensions()
{
	ImageFormatInfo formatInfo = (ImageFormatInfo)this.renderInfo.FormatInfo;

	if (formatInfo.failure == ImageFailure.None)
	{
		XImage xImage = null;
```

```
            try
            {
                  if (image.StreamBased)
                        xImage = XImage.FromStream(image.ImageStream);
                  else
                        xImage = XImage.FromFile(formatInfo.ImagePath);
            }
            catch
            {
...
```

DocumentElements.cs & ParagraphElements.cs

The fourth and fifth files that need to be changed are **DocumentElements.cs** and **ParagraphElements.cs**:
MigraDoc\code\MigraDoc.DocumentObjectModel\MigraDoc.DocumentObjectModel\DocumentElements.cs
MigraDoc\code\MigraDoc.DocumentObjectModel\MigraDoc.DocumentObjectModel\ ParagraphElements.cs

Change 1: Overload the AddImage method to accept a MemoryStream parameter

```
...
/// <summary>
/// Adds a new image to the collection.
/// </summary>
public Image AddImage(string name)
{
      Image image = new Image();
      image.Name = name;
      Add(image);
      return image;
}

/// <summary>
/// Adds a new image to the collection from a MemoryStream.
/// </summary>
/// <returns></returns>
public Image AddImage(MemoryStream stream)
{
      Image image = new Image();
      image.StreamBased = true;
      image.ImageStream = stream;
      image.Name = String.Empty;
      Add(image);
      return image;
}

/// <summary>
/// Adds a new text frame to the collection.
/// </summary>
public TextFrame AddTextFrame()
...
```

At this point, all the heavy lifting is done.

Now, you need to modify the various files within the MigraDoc.DocumentObjectModel namespace that have the `AddImage` method, overloading it where you want. I ended up doing *all* of the files. I'll list them here with a few notes.

`MigraDoc\code\MigraDoc.DocumentObjectModel\MigraDoc.DocumentObjectModel\Footnote.cs`
`MigraDoc\code\MigraDoc.DocumentObjectModel\MigraDoc.DocumentObjectModel\FormattedText.cs`
`MigraDoc\code\MigraDoc.DocumentObjectModel\MigraDoc.DocumentObjectModel\HeaderFooter.cs`
`MigraDoc\code\MigraDoc.DocumentObjectModel\MigraDoc.DocumentObjectModel\Hyperlink.cs`
`MigraDoc\code\MigraDoc.DocumentObjectModel\MigraDoc.DocumentObjectModel\Paragraph.cs`
`MigraDoc\code\MigraDoc.DocumentObjectModel\MigraDoc.DocumentObjectModel\Section.cs`
`MigraDoc\code\MigraDoc.DocumentObjectModel\MigraDoc.DocumentObjectModel.Shapes\TextFrame.cs`
`MigraDoc\code\MigraDoc.DocumentObjectModel\MigraDoc.DocumentObjectModel.Shapes.Charts\TextArea.cs`
`MigraDoc\code\MigraDoc.DocumentObjectModel\MigraDoc.DocumentObjectModel.Tables\Cell.cs`

Most often, you use the `AddImage` method in a `Paragraph` or within `FormattedText`, but I found that I needed it in the `HeaderFooter` as well as the `Cell`. So you'll have to overload the `AddImage` wherever you need it.

The overload is simple. If you look in the `Paragraph.cs` file (see path above), you'll see that the overload simply calls the overloaded `AddImage` from `ParagraphElements.cs` or `DocumentElements.cs`

Paragraph.cs – Change: Overload the `AddImage` method to accept a `MemoryStream` parameter

```
...
/// <summary>
/// Adds a new Image object
/// </summary>
public Image AddImage(string fileName)
{
      return this.Elements.AddImage(fileName);
}

/// <summary>
/// Adds a new Image to the paragraph from a MemoryStream.
/// </summary>
/// <returns></returns>
public Image AddImage(MemoryStream stream)
{
      return this.Elements.AddImage(stream);
}

/// <summary>
/// Adds a new Bookmark
/// </summary>
...
```

Do this for all the files listed above, depending on your needs.

That’s it!

Build it and use it.

So far this has worked for me. I hope it works for you ☺