using Mallenom;

using Mallenom.Imaging;

namespace Viscont.Core.Framework.ImageDataTransmission;

public record class ImageMetadata(

string ImageName,

ImageDataFormat ImageFormat,

int ImageWidth,

int ImageHeight,

string ImageFileType);

using Mallenom.Framework;

using Mallenom.Imaging;

using System;

namespace Viscont.Core.Framework.ImageDataTransmission;

public interface IImageDataWriter

{

IDisposable WriteImageToMemory(

Guid imageId,

ImageData imageDataReference);

}

using System;

using Mallenom.Framework;

using Mallenom.Imaging;

namespace Viscont.Core.Framework.ImageDataTransmission;

public interface IImageDataReader

{

void ReadImageFromMemory(

Guid imageId,

Reference<ImageData> reference);

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Viscont.Core.Framework.ImageDataTransmission;

public record class ImageMetadataModel(

string FileName,

int Width,

int Height,

string Format,

string FileFormat);

using System;

using System.IO.MemoryMappedFiles;

using Mallenom.Imaging;

namespace Viscont.Core.Framework.ImageDataTransmission;

public class ImageDataWriter : IImageDataWriter

{

#region Implementation

public IDisposable WriteImageToMemory(

Guid guid,

ImageData imageData)

{

var imageSize = ImageDataLayout.GetRequiredCapacity(imageData.Format, imageData.Width, imageData.Height);

var memoryMappedFile = MemoryMappedFile.CreateNew(guid.ToString(), imageSize);

using var writer = memoryMappedFile.CreateViewAccessor(0, imageSize);

WriteToMemory(imageData, writer);

return memoryMappedFile;

}

#endregion

#region Methods

private static unsafe void WriteToMemory(

ImageData imageData,

MemoryMappedViewAccessor writer)

{

byte\* ptr = null;

writer.SafeMemoryMappedViewHandle.AcquirePointer(ref ptr);

try

{

var layout = ImageDataLayout.Create(

(IntPtr)ptr,

imageData.Format,

imageData.Width,

imageData.Height);

using var data = new ImageData(

layout.Slice0,

layout.Slice1,

layout.Slice2,

imageData.Width,

imageData.Height,

imageData.Format);

ColorSpaceConverter.Convert(imageData, data);

}

finally

{

writer.SafeMemoryMappedViewHandle.ReleasePointer();

}

}

#endregion

}

using System;

using System.IO.MemoryMappedFiles;

using Mallenom.Framework;

using Mallenom.Imaging;

namespace Viscont.Core.Framework.ImageDataTransmission;

public class ImageDataReader : IImageDataReader

{

#region Data

private readonly IImageDataAllocator \_imageDataAllocator;

#endregion

#region .ctor

public ImageDataReader(IImageDataAllocator imageDataAllocator)

{

\_imageDataAllocator = imageDataAllocator

?? throw new ArgumentNullException(nameof(imageDataAllocator));

}

#endregion

#region Implementation

public void ReadImageFromMemory(

Guid imageId,

Reference<ImageData> reference)

{

var imageData = reference.Value;

int sizeImage = ImageDataLayout.GetRequiredCapacity(imageData!.Format,

imageData.Width, imageData.Height);

using var sharedMemory = OperatingSystem.IsWindows()

? MemoryMappedFile.OpenExisting(imageId.ToString("N"))

: throw new PlatformNotSupportedException();

using var reader = sharedMemory.CreateViewAccessor(0, sizeImage, MemoryMappedFileAccess.Read);

ReadFromMemory(reader, reference);

}

#endregion

#region Methods

private unsafe void ReadFromMemory(

MemoryMappedViewAccessor reader,

Reference<ImageData> reference)

{

byte\* ptr = null;

reader.SafeMemoryMappedViewHandle.AcquirePointer(ref ptr);

try

{

if(!\_imageDataAllocator.TryAllocate(

reference,

reference.Value!.Format,

reference.Value!.Width,

reference.Value!.Height))

{

throw new Exception("\_imageDataAllocator.TryAllocate return false");

}

try

{

var layout = ImageDataLayout.Create(

(IntPtr)ptr,

reference.Value!.Format,

reference.Value!.Width,

reference.Value!.Height);

using var src = new ImageData(

layout.Slice0,

layout.Slice1,

layout.Slice2,

reference.Value!.Width,

reference.Value!.Height,

reference.Value!.Format);

ColorSpaceConverter.Convert(src, reference.Value!);

}

catch

{

reference.UnreferenceValue();

throw;

}

}

finally

{

reader.SafeMemoryMappedViewHandle.ReleasePointer();

}

}

#endregion

}