Calling Functions on Unity-Application embedded in Winforms-Application [duplicate]

This question already has an answer here:

- Send message from one program to another in Unity 2 answers
- Embed Unity3D app inside WPF application 1 answer

I am currently developing a simple prototype for an editor. The editor will use WinForms (or WPF, if possible) to provide the main user interface and will also embed a Unity 2017 standalone application to visualise data and provide additional control elements (e.g. zoom, rotate, scroll, ...).

Thanks to this nice post below, getting an embedded Unity application to work within a WinForms-application was shockingly easy.

https://forum.unity.com/threads/unity-3d-within-windows-application-enviroment.236213/

Also, there is a simple example application, which you may access here:

Example.zip

Unfortunately, neither the example, nor any posts I could find answered a very basic question: *how do you pass data (or call methods) from your WinForms-application in your embedded Unity-application (and vice versa)?*

Is it possible for your WinForms-application to simply call MonoBehaviour-scripts or static methods in your Unity-application? If so, how? If not, what would be a good workaround? And how could the Unity-to-WinForms communication work in return?

Update:

Used the duplicate-pages mentioned by Programmer (link) to implement a solution, which uses named pipes for communication between the WinForms- and Unity-application.

Both applications use BackgroundWorkers, the WinForms-application acts as server (since it is started first and needs an active connection-listener, before the client is started), while the embedded Unity-application acts as client.

Unfortunately, the Unity-application throws a NotImplementedException, stating "ACL is not supported in Mono" when creating the NamedPipeClientStream (tested with Unity 2017.3 and Net 2.0 (not the Net 2.0 subset)). This exception has already been reported in some comments in the post mentioned above, but its unclear, if it has been solved. The proposed solutions of "make sure, that the server is up and running before the client tries to connect" and "start it in admin mode" have been tried, but failed so far.

Solution:

After some more testing, it became clear, that the "ACL is not supported in Mono" exception was caused the TokenImpersonationLevel-parameter used when creating the NamedPipeClientStream-instance. Changing it to TokenImpersonationLevel.None solved the issue.

Here the code used by the WinForms-application, which acts as a named pipe server. Make sure, this script is executed, BEFORE the Unity-application client tries to connect! Also, make sure, that you have build and released the Unity-application before you start the server. Place the Unity executable of your Unity-application in the WinForms-applications folder and name it "Child.exe".

```
using System;
using System.ComponentModel;
using System.Runtime.InteropServices;
using System.Threading;
using System.Windows.Forms;
using System.Diagnostics;
using System.IO.Pipes;

namespace Container
{
    public partial class MainForm : Form
    {
        [DllImport("User32.dll")]
        static extern bool MoveWindow(IntPtr handle, int x, int y, int width, int
height, bool redraw);

        internal delegate int WindowEnumProc(IntPtr hwnd, IntPtr lparam);
        [DllImport("user32.dll")]
        internal static extern bool EnumChildWindows(IntPtr hwnd, WindowEnumProc
func, IntPtr lParam);

        [DllImport("user32.dll")]
        static extern int SendMessage(IntPtr hWnd, int msg, IntPtr wParam, IntPtr
lParam);

        /// <summary>
        /// A Delegate for the Update Log Method.
        /// </summary>
        /// <param name="text">The Text to log.</param>
        private delegate void UpdateLogCallback(string text);

        /// <summary>
        /// The Unity Application Process.
        /// </summary>
        private Process process;

        /// <summary>
        /// The Unity Application Window Handle.
        /// </summary>
        private IntPtr unityHWND = IntPtr.Zero;

        private const int WM_ACTIVATE = 0x0006;
        private readonly IntPtr WA_ACTIVE = new IntPtr(1);
        private readonly IntPtr WA_INACTIVE = new IntPtr(0);
```

```
        /// <summary>
        /// The Background Worker, which will send and receive Data.
        /// </summary>
        private BackgroundWorker backgroundWorker;

        /// <summary>
        /// A Named Pipe Stream, acting as the Server for Communication between this
Application and the Unity Application.
        /// </summary>
        private NamedPipeServerStream namedPipeServerStream;



        public MainForm()
        {
            InitializeComponent();

            try
            {
                //Create Server Instance
                namedPipeServerStream = new
NamedPipeServerStream("NamedPipeExample", PipeDirection.InOut, 1);

                //Start Background Worker
                backgroundWorker = new BackgroundWorker();
                backgroundWorker.DoWork += new
DoWorkEventHandler(backgroundWorker_DoWork);
                backgroundWorker.WorkerReportsProgress = true;

                backgroundWorker.RunWorkerAsync();

                //Start embedded Unity Application
                process = new Process();
                process.StartInfo.FileName = Application.StartupPath +
"\\Child.exe";
                process.StartInfo.Arguments = "-parentHWND " +
splitContainer.Panel1.Handle.ToInt32() + " " + Environment.CommandLine;
                process.StartInfo.UseShellExecute = true;
                process.StartInfo.CreateNoWindow = true;

                process.Start();
                process.WaitForInputIdle();

                //Embed Unity Application into this Application
                EnumChildWindows(splitContainer.Panel1.Handle, WindowEnum,
IntPtr.Zero);

                //Wait for a Client to connect
                namedPipeServerStream.WaitForConnection();
            }
            catch (Exception ex)
            {
                throw ex;
            }

        }
```

```
        /// <summary>
        /// Activates the Unity Window.
        /// </summary>
        private void ActivateUnityWindow()
        {
            SendMessage(unityHWND, WM_ACTIVATE, WA_ACTIVE, IntPtr.Zero);
        }

        /// <summary>
        /// Deactivates the Unity Window.
        /// </summary>
        private void DeactivateUnityWindow()
        {
            SendMessage(unityHWND, WM_ACTIVATE, WA_INACTIVE, IntPtr.Zero);
        }

        private int WindowEnum(IntPtr hwnd, IntPtr lparam)
        {
            unityHWND = hwnd;
            ActivateUnityWindow();
            return 0;
        }

        private void panel1_Resize(object sender, EventArgs e)
        {
            MoveWindow(unityHWND, 0, 0, splitContainer.Panel1.Width,
splitContainer.Panel1.Height, true);
            ActivateUnityWindow();
        }

        /// <summary>
        /// Called, when this Application is closed. Tries to close the Unity
Application and the Named Pipe as well.
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void Form1_FormClosed(object sender, FormClosedEventArgs e)
        {
            try
            {
                //Close Connection
                namedPipeServerStream.Close();

                //Kill the Unity Application
                process.CloseMainWindow();

                Thread.Sleep(1000);

                while (process.HasExited == false)
                {
                    process.Kill();
                }
            }
            catch (Exception ex)
            {
```

```
                throw ex;
            }
        }

        private void MainForm_Activated(object sender, EventArgs e)
        {
            ActivateUnityWindow();
        }

        private void MainForm_Deactivate(object sender, EventArgs e)
        {
            DeactivateUnityWindow();
        }

        /// <summary>
        /// A simple Background Worker, which sends Data to the Client via a Named
Pipe and receives a Response afterwards.
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void backgroundWorker_DoWork(object sender, DoWorkEventArgs e)
        {
            //Init
            UpdateLogCallback updateLog = new UpdateLogCallback(UpdateLog);
            string dataFromClient = null;

            try
            {
                //Don't pass until a Connection has been established
                while (namedPipeServerStream.IsConnected == false)
                {
                    Thread.Sleep(100);
                }

                //Created stream for reading and writing
                StreamString serverStream = new StreamString(namedPipeServerStream);

                //Send to Client and receive Response (pause using Thread.Sleep for
demonstration Purposes)
                Invoke(updateLog, new object[] { "Send Data to Client: " +
serverStream.WriteString("A Message from Server.") + " Bytes." });
                Thread.Sleep(1000);
                dataFromClient = serverStream.ReadString();
                Invoke(updateLog, new object[] { "Received Data from Server: " +
dataFromClient });

                Thread.Sleep(1000);

                Invoke(updateLog, new object[] { "Send Data to Client: " +
serverStream.WriteString("A small Message from Server.") + " Bytes." });
                Thread.Sleep(1000);
                dataFromClient = serverStream.ReadString();
                Invoke(updateLog, new object[] { "Received Data from Server: " +
dataFromClient });

                Thread.Sleep(1000);
```

```
                Invoke(updateLog, new object[] { "Send Data to Client: " +
serverStream.WriteString("Another Message from Server.") + " Bytes." });
                Thread.Sleep(1000);
                dataFromClient = serverStream.ReadString();
                Invoke(updateLog, new object[] { "Received Data from Server: " +
dataFromClient });

                Thread.Sleep(1000);

                Invoke(updateLog, new object[] { "Send Data to Client: " +
serverStream.WriteString("The final Message from Server.") + " Bytes." });
                Thread.Sleep(1000);
                dataFromClient = serverStream.ReadString();
                Invoke(updateLog, new object[] { "Received Data from Server: " +
dataFromClient });
            }
            catch(Exception ex)
            {
                //Handle usual Communication Exceptions here - just logging here for
demonstration and debugging Purposes
                Invoke(updateLog, new object[] { ex.Message });
            }
        }

        /// <summary>
        /// A simple Method, which writes Text into a Console / Log. Will be invoked
by the Background Worker, since WinForms are not Thread-safe and will crash, if
accessed directly by a non-main-Thread.
        /// </summary>
        /// <param name="text">The Text to log.</param>
        private void UpdateLog(string text)
        {
            lock (richTextBox_Console)
            {
                Console.WriteLine(text);
                richTextBox_Console.AppendText(Environment.NewLine + text);
            }
        }
    }
}
```

Attach this code to a GameObject within your Unity-application. Also make sure to reference a GameObject with a TextMeshProUGUI-component (TextMeshPro-Asset, you'll find it in your Asset Store) to the 'textObject'-member, so the application doesn't crash and you can see some debug information. Also (as stated above) make sure you build and release your Unity-application, name it "Child.exe" and put it in the same folder as your WinForms-application.

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;
using System;
using System.IO.Pipes;
using System.Security.Principal;
using Assets;
```

```
using System.ComponentModel;
using TMPro;



/// <summary>
/// A simple Example Project, which demonstrates Communication between WinForms-
Applications and embedded Unity Engine Applications via Named Pipes.
///
/// This Code (Unity) is considered as the Client, which will receive Data from the
WinForms-Server and send a Response in Return.
/// </summary>
public class SendAndReceive : MonoBehaviour
{
    /// <summary>
    /// A GameObject with an attached Text-Component, which serves as a simple
Console.
    /// </summary>
    public GameObject textObject;

    /// <summary>
    /// The Background Worker, which will send and receive Data.
    /// </summary>
    private BackgroundWorker backgroundWorker;

    /// <summary>
    /// A Buffer needed to temporarely save Text, which will be shown in the
Console.
    /// </summary>
    private string textBuffer = "";



    /// <summary>
    /// Use this for initialization.
    /// </summary>
    void Start ()
    {
        //Init the Background Worker to send and receive Data
        this.backgroundWorker = new BackgroundWorker();
        this.backgroundWorker.DoWork += new
DoWorkEventHandler(backgroundWorker_DoWork);
        this.backgroundWorker.WorkerReportsProgress = true;
        this.backgroundWorker.RunWorkerAsync();
    }

    /// <summary>
    /// Update is called once per frame.
    /// </summary>
    void Update ()
    {
        //Update the Console for debugging Purposes
        lock (textBuffer)
        {
            if (string.IsNullOrEmpty(textBuffer) == false)
            {
                textObject.GetComponent<TextMeshProUGUI>().text =
```

```
textObject.GetComponent<TextMeshProUGUI>().text + Environment.NewLine + textBuffer;
                textBuffer = "";
            }
        }
    }

    /// <summary>
    /// A simple Background Worker, which receives Data from the Server via a Named
Pipe and sends a Response afterwards.
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void backgroundWorker_DoWork(object sender, DoWorkEventArgs e)
    {
        try
        {
            //Init
            NamedPipeClientStream client = null;
            string dataFromServer = null;

            //Create Client Instance
            client = new NamedPipeClientStream(".", "NamedPipeExample",
PipeDirection.InOut, PipeOptions.None, TokenImpersonationLevel.None);
            updateTextBuffer("Initialized Client");

            //Connect to Server
            client.Connect();
            updateTextBuffer("Connected to Server");

            //Created stream for Reading and Writing
            StreamString clientStream = new StreamString(client);

            //Read from Server and send Response (flush in between to clear the
Buffer and fix some strange Issues I couldn't really explain, sorry)
            dataFromServer = clientStream.ReadString();
            updateTextBuffer("Received Data from Server: " + dataFromServer);
            client.Flush();
            updateTextBuffer("Sent Data back to Server: " +
clientStream.WriteString("Some data from client.") + " Bytes.");

            dataFromServer = clientStream.ReadString();
            updateTextBuffer("Received Data from Server: " + dataFromServer);
            client.Flush();
            updateTextBuffer("Sent Data back to Server: " +
clientStream.WriteString("Some more data from client.") + " Bytes.");

            dataFromServer = clientStream.ReadString();
            updateTextBuffer("Received Data from Server: " + dataFromServer);
            client.Flush();
            updateTextBuffer("Sent Data back to Server: " +
clientStream.WriteString("A lot of more data from client.") + " Bytes.");

            dataFromServer = clientStream.ReadString();
            updateTextBuffer("Received Data from Server: " + dataFromServer);
            client.Flush();
            updateTextBuffer("Sent Data back to Server: " +
```

```
clientStream.WriteString("Clients final message.") + " Bytes.");

            //Close client
            client.Close();
            updateTextBuffer("Done");
        }
        catch (Exception ex)
        {
            //Handle usual Communication Exceptions here - just logging here for
demonstration and debugging Purposes
            updateTextBuffer(ex.Message + Environment.NewLine +
ex.StackTrace.ToString() + Environment.NewLine + "Last Message was: " + textBuffer);
        }
    }

    /// <summary>
    /// A Buffer, which allows the Background Worker to save Texts, which may be
written into a Log or Console by the Update-Loop
    /// </summary>
    /// <param name="text">The Text to save.</param>
    private void updateTextBuffer(string text)
    {
        lock (textBuffer)
        {
            if (string.IsNullOrEmpty(textBuffer))
            {
                textBuffer = text;
            }
            else
            {
                textBuffer = textBuffer + Environment.NewLine + text;
            }
        }
    }
}
```

Also, both scripts need an additional class, which encapsulates the pipe stream, so sending and receiving text becomes much easier.

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;

namespace Assets
{
    /// <summary>
    /// Simple Wrapper to write / read Data to / from a Named Pipe Stream.
    ///
    /// Code based on:
    /// https://stackoverflow.com/questions/43062782/send-message-from-one-program-
to-another-in-unity
    /// </summary>
    public class StreamString
    {
        private Stream ioStream;
        private UnicodeEncoding streamEncoding;

        public StreamString(Stream ioStream)
        {
            this.ioStream = ioStream;
            streamEncoding = new UnicodeEncoding();
        }

        public string ReadString()
        {
            int len = 0;

            len = ioStream.ReadByte() * 256;
            len += ioStream.ReadByte();
            byte[] inBuffer = new byte[len];
            ioStream.Read(inBuffer, 0, len);

            return streamEncoding.GetString(inBuffer);
        }

        public int WriteString(string outString)
        {
            byte[] outBuffer = streamEncoding.GetBytes(outString);
            int len = outBuffer.Length;
            if (len > UInt16.MaxValue)
            {
                len = (int)UInt16.MaxValue;
            }
            ioStream.WriteByte((byte)(len / 256));
            ioStream.WriteByte((byte)(len & 255));
            ioStream.Write(outBuffer, 0, len);
            ioStream.Flush();

            return outBuffer.Length + 2;
        }
    }
}
```

If you read the post down to this point: thank you :) I hope it will help you on your journey to a successfull developer!

Final result of my prototype: