Simio
Forward Thinking

# Simio API Note: Simio API Helper

Last Update: 13 June 2020

## Contents

Simio
Forward Thinking

# Overview

This API Note describes a utility that can:

1. Examine/test the DLLs that are used by the APIs.
2. Construct sample “headless” folders
3. Test the operation of the headless methods.

Additionally, there are sample projects included. The organization of the build file is described in the section “Using the Simio API Helper Code”

This code and documentation are intended to be distributed via a GitHub repository using Simio’s GitHub conventions. The reasons for using GitHub include:

1. Employing a standard mechanism for distributing API notes
2. Allowing the API information to evolve
3. Allowing the Simio community to engage in the production of Simio API examples

If you have suggestions for improving this example you can either:

- Send your ideas to Simio Support (support@Simio.com). Make sure you reference which GitHub repository you are commenting on.
- Ask for permission to contribute directly to this GitHub repository.

Simio
Forward Thinking

# Using the API Helper.

The Simio API Tester is not part of the Simio product. Rather, it is a demonstration and test tool that was built to help debug problems related to using the Simio API, as well as to demonstrate how to use the Simio API.

How much benefit you derive will depend upon your programming knowledge, but anyone attempting to use the API should find it useful.

You can use this tool to verify that your DLLs are:

1. In the right place
2. Accessible
3. Implementing the correct Interfaces
4. Referencing other Assemblies correctly

And in the current version you can also use it to:

1. Prepare your environment for running Simio in a “headless” (without the UI) mode.
2. Run a Simio model headless to
    a. Run experiments, and/or
    b. Run a Plan.

Note: You can only run a Plan if you have the Simio RPS edition (or the Personal Edition with a small demonstration model).

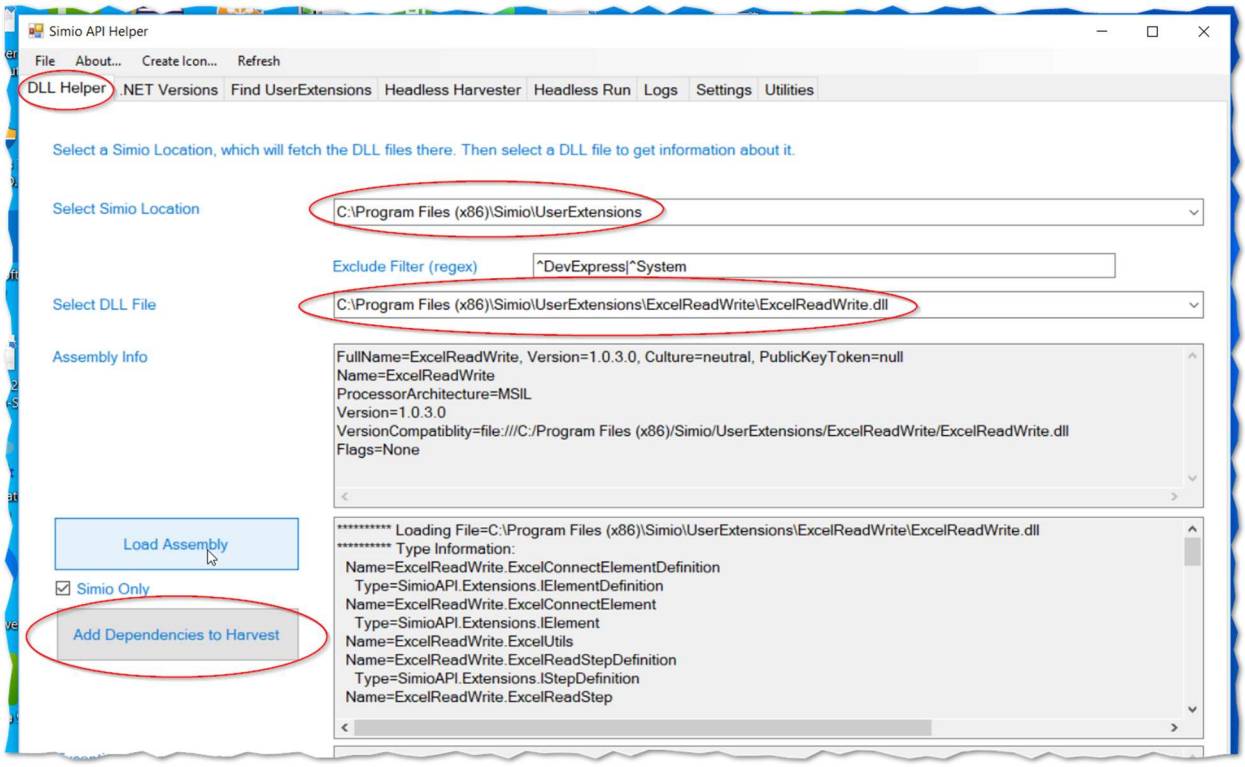
The UI architecture is a simple WinForms tabs, and the sections below are organized according to those tabs.

## Tab DLL Helper

This tab is used to examine and load DLL assemblies.

The top drop-down displays the locations where Simio DLLs can be found, and the next drop-down then shows the DLL files within that location. The Exclude filter can be used to reduce the number of DLL files displayed.

Once a file is chosen, general information about the contents of the DLL is shown, along with the definitions found within the file. If you wish to see only Simio information, check the “Simio Only” checkbox.



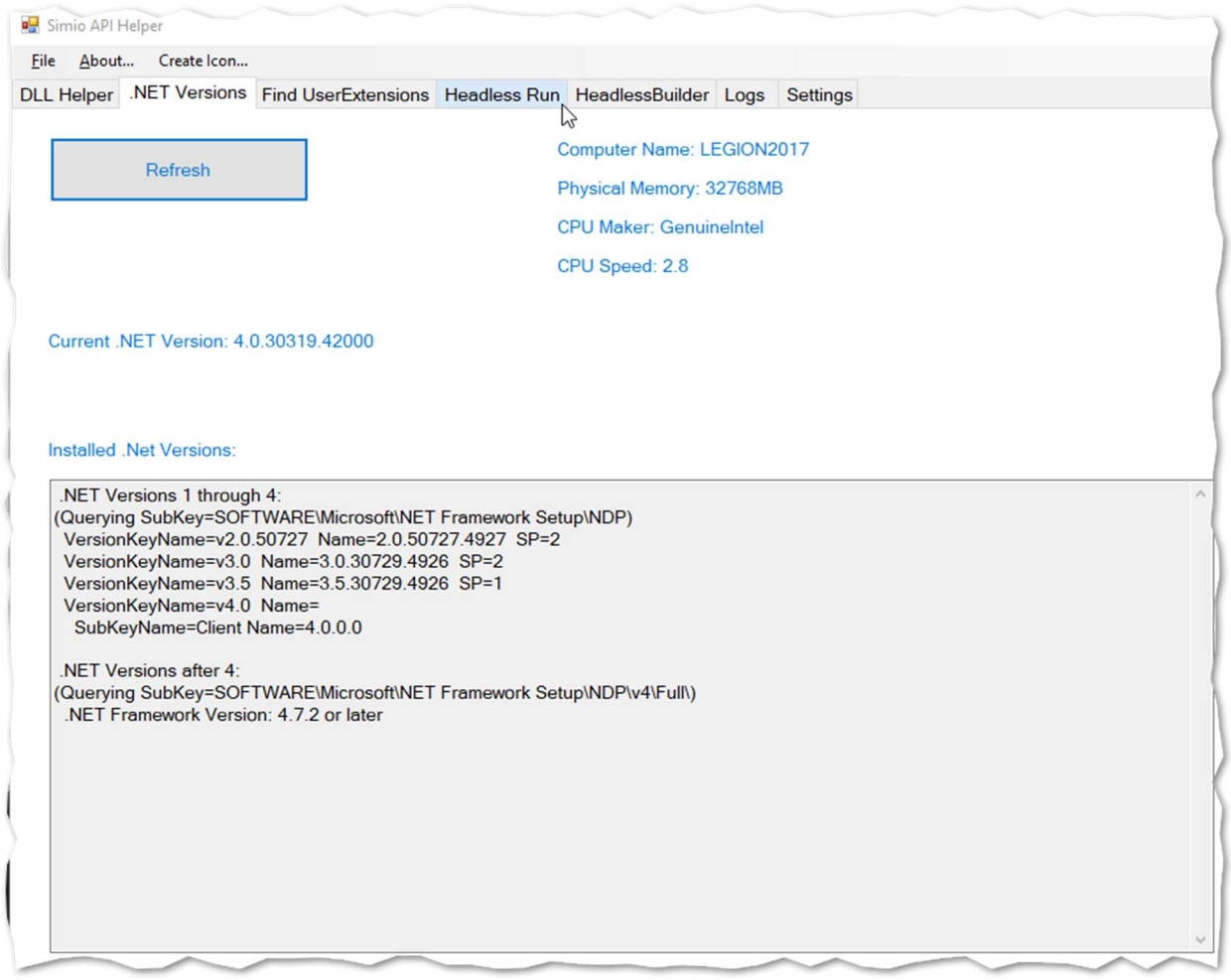

Using the “Load Assembly” you can get information about the Interfaces as well as DLL dependencies.

Using the “Add Dependency to Harvest” button, you can make sure that the DLLs upon which this DLL is dependent are in the target harvest folder.

Simio
Forward Thinking

## Tab .Net Versions

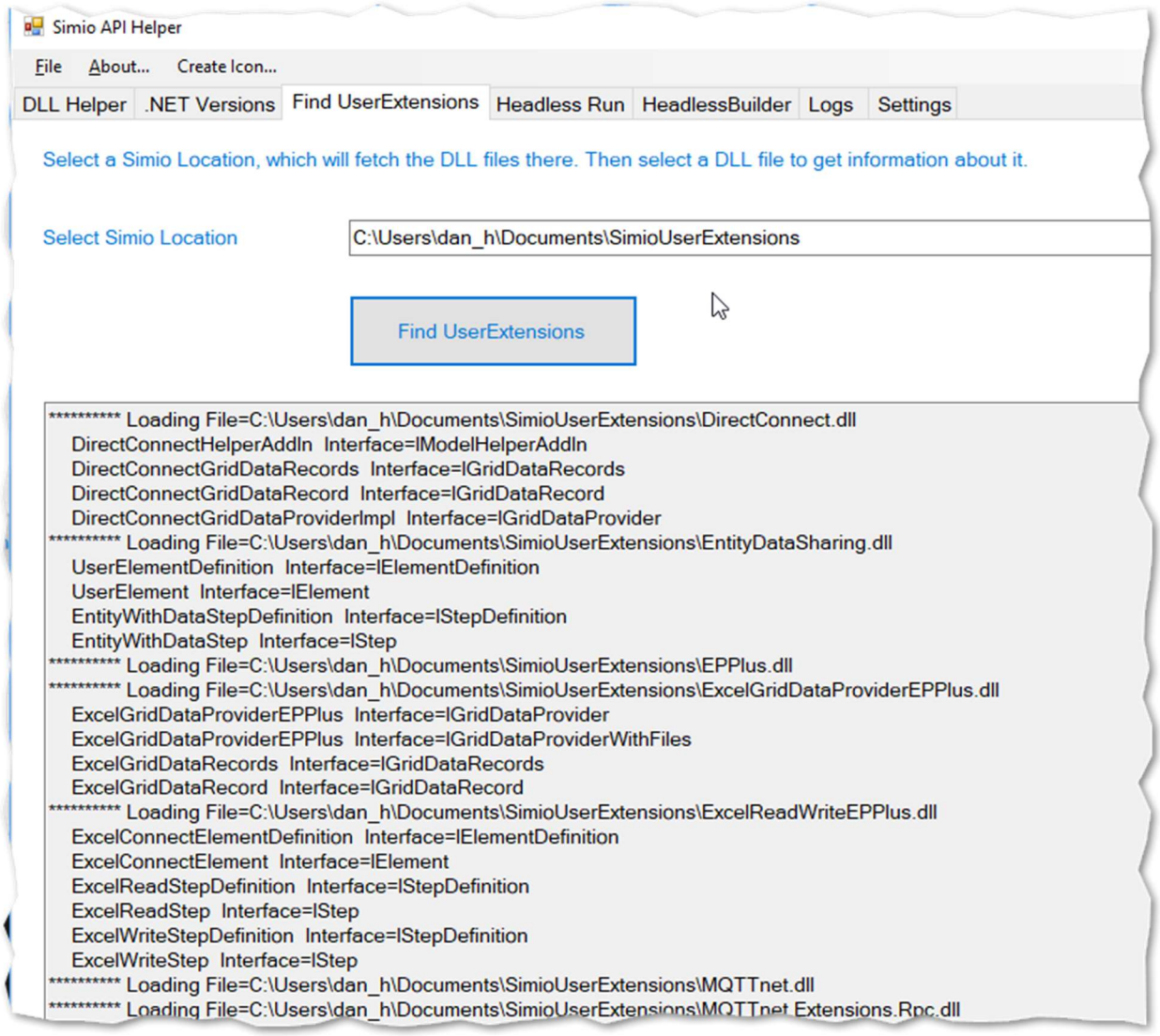
This tab shows general information about the computer that it is run on, as well as the .NET versions that are installed.



## Tab: Find User Extensions

Given a starting point (which is generally the default SimioUserExtensions folder beneath a user's Documents folder), this tab is used to locate all the assemblies (DLLs) containing Simio interfaces.

All the Simio interfaces within each DLL are listed. For example, in the listing below, there is a DLL called ExcelReadWriteEPPLus.dll that contains interfaces such as IStepDefinition and IStep, which identifies it as a User Step extension.



Simio
Forward Thinking

# Simio Automated (Headless) Workflow and Recommendations

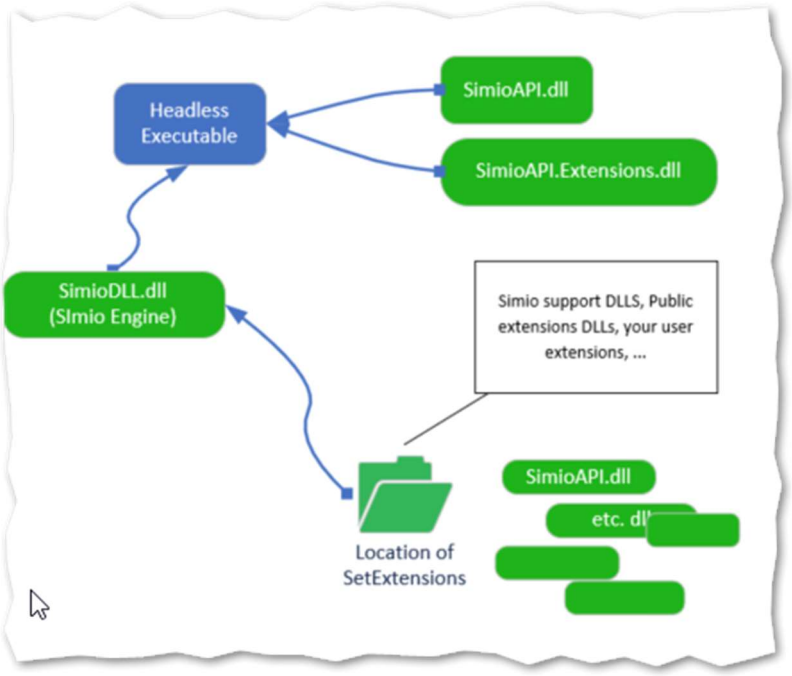
When you are running Simio from the desktop version you are running the Simio "Engine" with the standard bolted-on Simio UI. This is the most popular and recommended configuration for designing, building, and interacting with your model. However, there are several use cases for running your model without the Simio UI, and this is where the automation workflow ("headless") configuration comes in.

As Simio is designed to be data driven, a common scenario is to have a headless application awaiting the arrival of new data, and then processing the Simio project (to run experiments or schedule a plan) and then storing the resulting data. Simio has a range of APIs to assist this with this headless mode.

Achieving success when building a headless application often depends upon selecting and using the correct components. These binary components are called "assemblies", and consist of libraries (files with DLL extensions and/or executables, which are files with EXE extensions.

The task can appear daunting because Simio is very modularized and determining which DLLs to use can be a confusing task. It is further challenging because the executable that you build will reference Simio API DLLs to load the Simio Engine, and then the engine will - in turn -require more DLLs dependent upon other requirements, such as the type of User Extensions that you are using.

This section will provide you with some background knowledge, recommendations, and debugging techniques to help sort this out.



Simio
Forward Thinking

Let's start with a recommendation: when you wish to have a headless configuration, it is wise to collect or "harvest" all the Simio components that you need into a dedicated folder, such as c:/(test)/SimioHeadlessTest.

In theory you *could* use the Simio installation folder, but this would mean that every time you update Simio you will have to re-test your headless application. And perhaps more importantly, when the Simio Engine is running it looks for DLLs (such as custom user extensions) in many places. ***When you are using headless this searching does not occur!*** So, if you unwisely decided to do this, you would have to move all the DLLs that your application uses into the Simio installation folder, causing unnecessary clutter, duplication, and confusion.

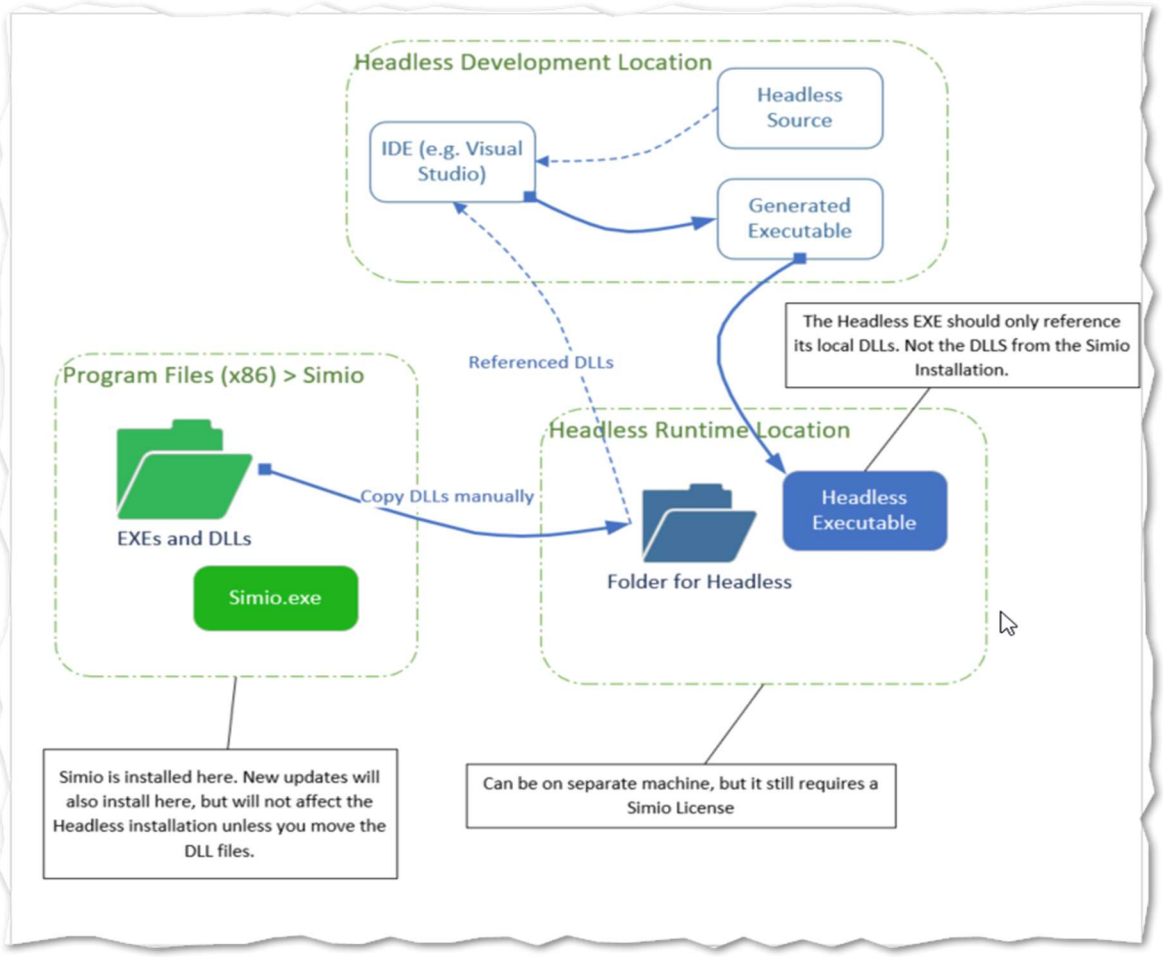
Note that the Simio API Helper has a utility that can help you harvest the most likely needed DLLs from your Simio installation and put copies in a folder of your choice. This function is found under the tab named "Headless Harvester".

This has the advantage of protecting your application (which are often production oriented) from updates or upgrades to the Simio software, and you have all your dependent DLLs in a convenient package for backing up. The downside of course is that when you do update Simio to a new version you must decide if you want your headless system to use the new update, and then re-harvest (and re-test) your headless system.

When you are building (and testing) a custom headless application (e.g. with Visual Studio) your executable will generally "look" for its supporting DLLs that are in the same folder as the executable.

The diagram below illustrates this workflow.

Simio
Forward Thinking



Simio
Forward Thinking

## The Code for a Headless Operation

The actual code is conceptually simple and generally of the form

1. Look for a reason to run, such as new data arriving, or a new Simio project file.
2. Set the extensions folder (which instructs the code where to look for the DLLs)
3. Load the Model (and check for errors)
4. Run the Experiment or Plan for the model.
5. Optionally Save the results

BTW: The most common error when using the headless mode is to omit Step 2, which instructs the Simio Engine where to find its dependent DLLs

Simio
Forward Thinking

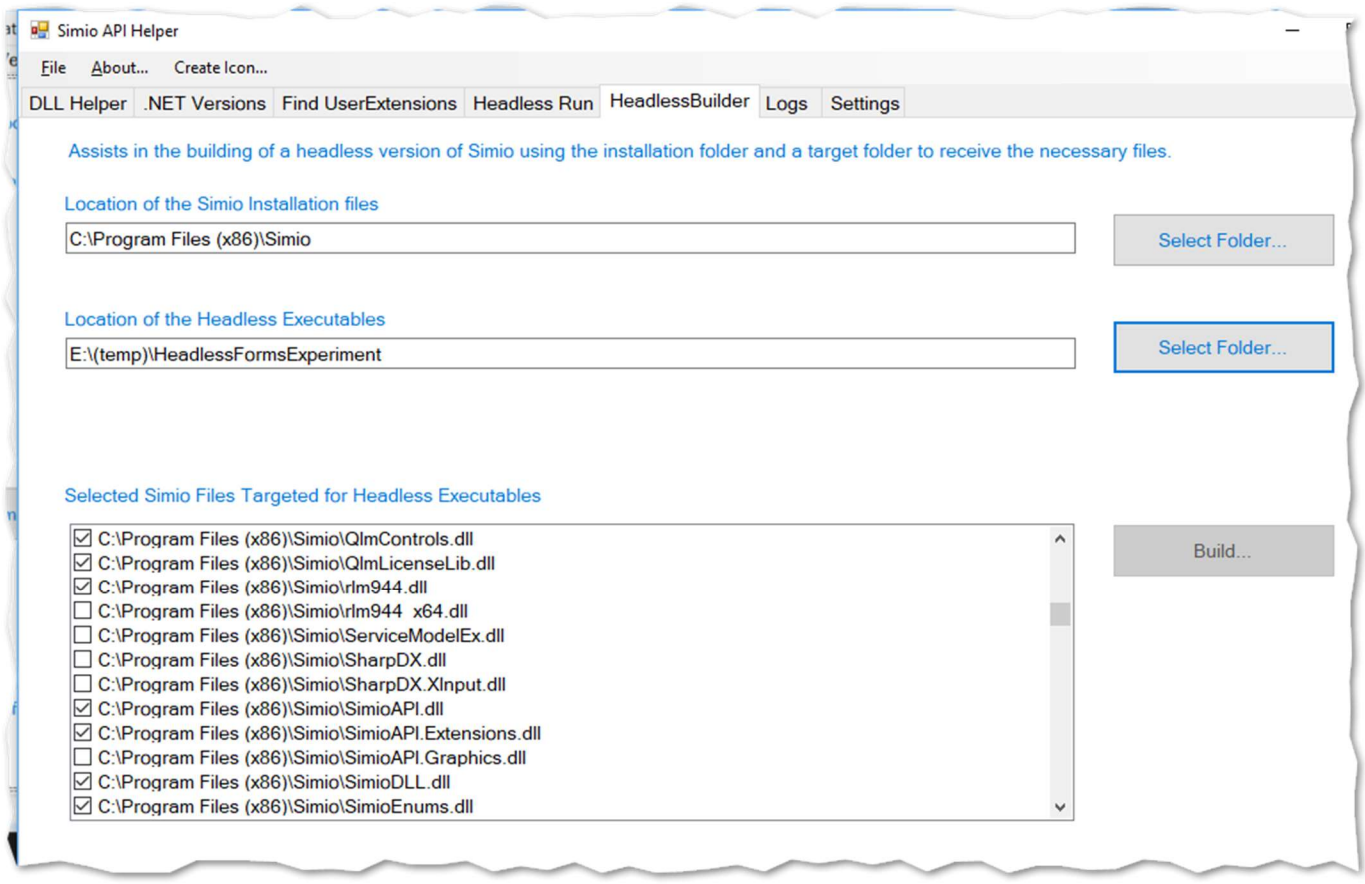
## Tabs: Headless Harvester and Headless Run

These tabs can help you set up, test, and run your Simio headless application.

The Harvester tab permits you to take (harvest) all the files that your headless operation needs from where you installed Simio, and places all the non-UI files into a folder of your choice.

Once you have chosen the Simio installation location and the target location it will show you (in the checkbox list) all the files that will be moved. You can selectively uncheck files that you are sure you don't need, or just leave the recommended files checked; in general, extra files won't hurt.

Pressing the "Start Harvest" will move all the checked files to your target location.



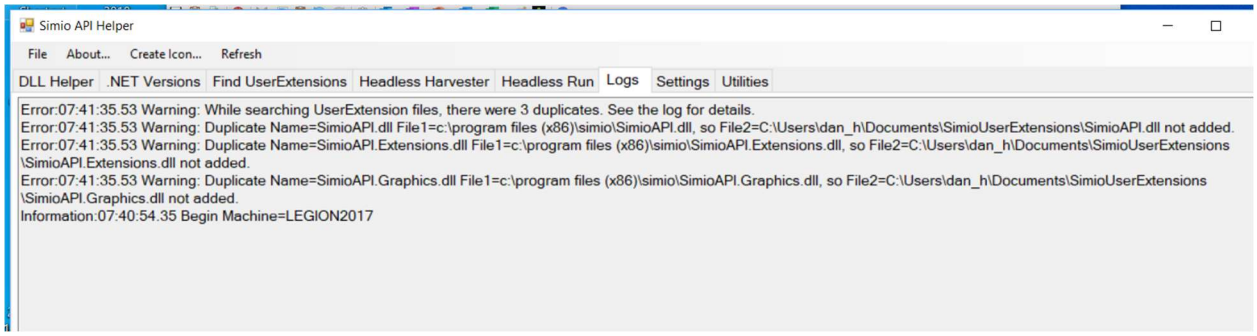

The Headless Run provides a few options.

Simio
Forward Thinking

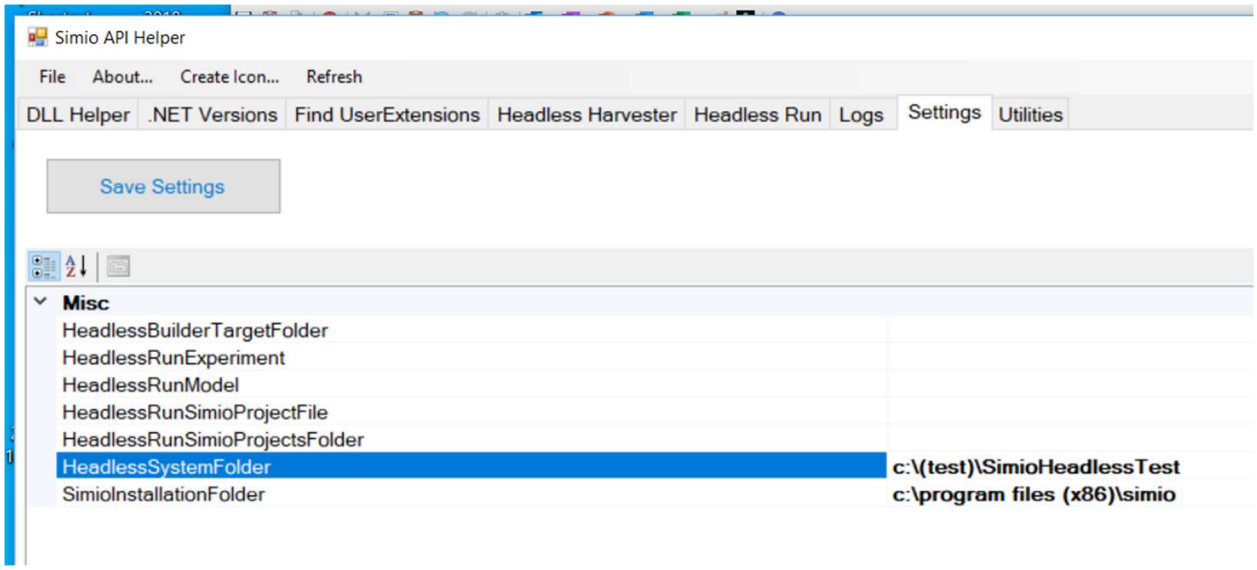
## Tabs: Logs and Settings

There are two utility tabs to assist with running and debugging.

The tab Logs shows messages that the SimioApiHelper generates. These are made by a utility class called Loggerton, which generates in-memory logging. Prudent logging will help you debug your program.




The tab Settings allows the program to remember your configuration, so you don't have to reenter each time you run. For example, the folder where Simio is installed is stored here, as well as the folder that you choose as the target for harvesting the assemblies.



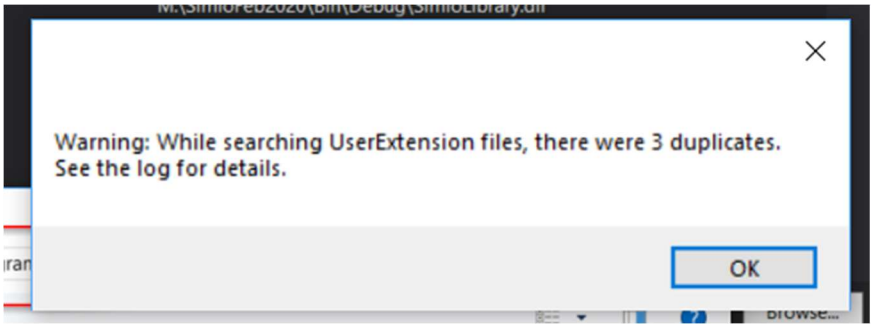
Simio
Forward Thinking

# Running Simio API Helper

The files built from SimioApiHelper are manually moved to the top-level Executables folder. There you will find the SimioApiHelper executable (SimioApiHelper.exe) as well as all of its dependent assemblies (*.DLL) and its configuration file.

You can start is by double clicking on SImioApiHelper.exe.

When starting, you may see errors like this:



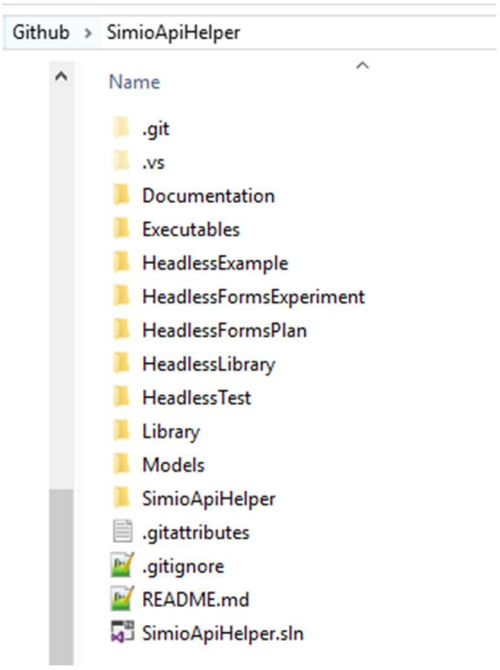

This is from the harvesting refresh, and it means that there were DLLs with the same name that exist in multiple folders of the Simio installation. It is mostly harmless, but you should look at the Logs tab and investigate why there were duplicates.

Simio
Forward Thinking

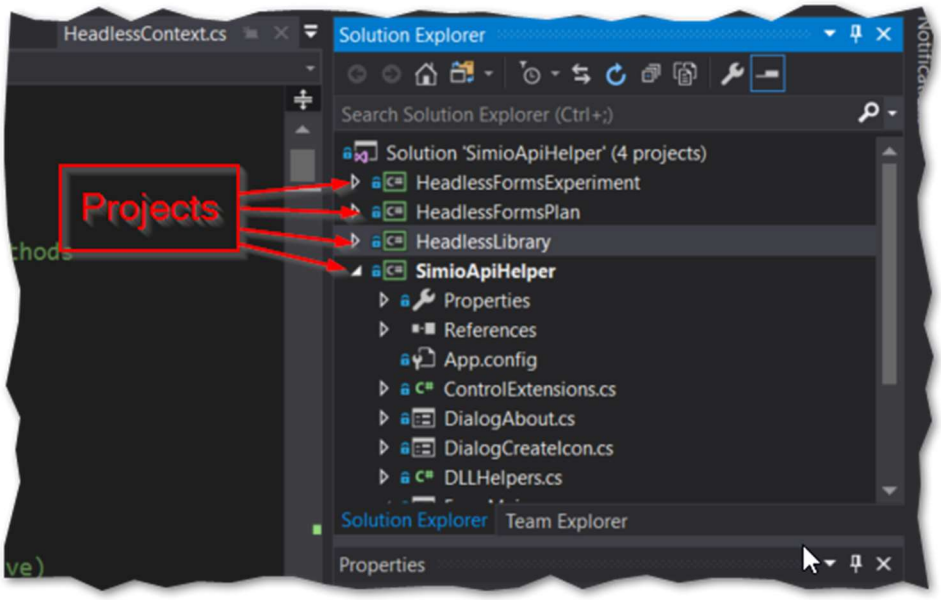
# Using the Simio API Helper Code

Although you can simply run the Simio API Helper by going to the Executables file and double clicking on SimioApiHelper.exe, you can also run and debug the code yourself using Visual Studio. A familiarity with .NET, Visual Studio, and the C# language is assumed.

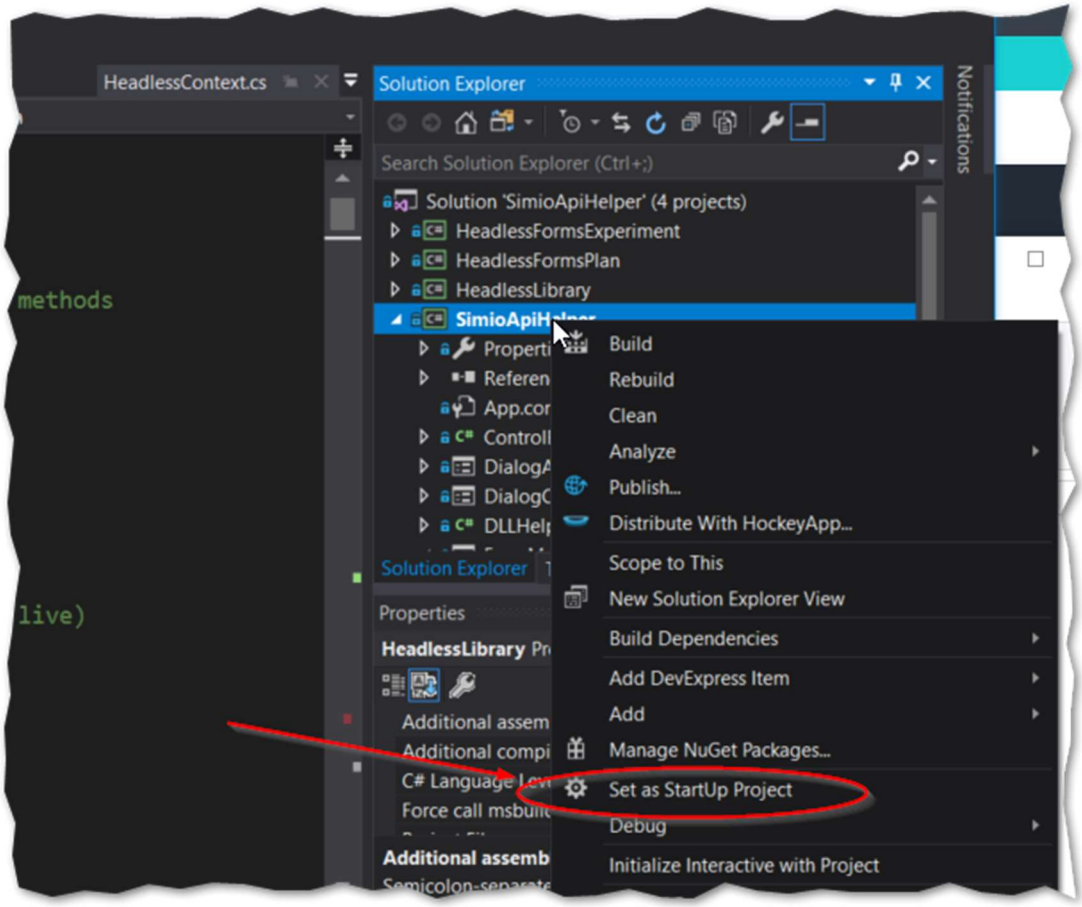
After cloning the repository from GitHub, the entire solution can be loaded in Visual Studio (version 2017 or 2019) via the top-level Visual Studio solution file SimioApiHelper.sln.




The solution file contains several example projects, including the SimioApiHelper.




You can switch from one project to another by select a project, right-clicking and choosing “Set as Startup Project”. Once done, that project will appear bolder than the others.




At this point, clicking on Start will cause that project to execute.

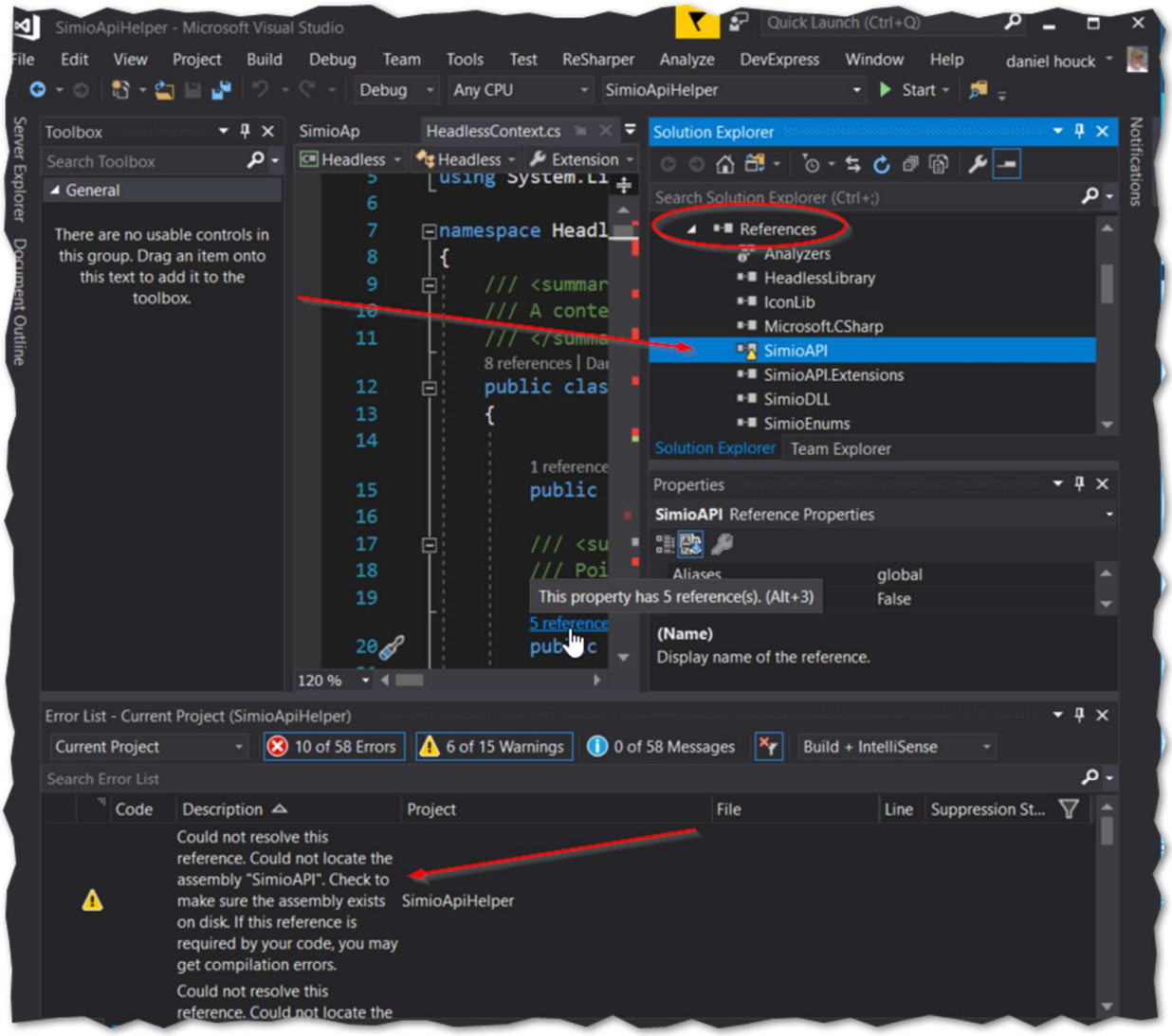
When you first try to run, you may get an error indicating that one of more of the Simio DLLs cannot be located.

You will need to locate these file in the folder where Simio is installed.

So first, delete the files from the projects References folder (you will get a warning dialog that the Selected items will be removed).

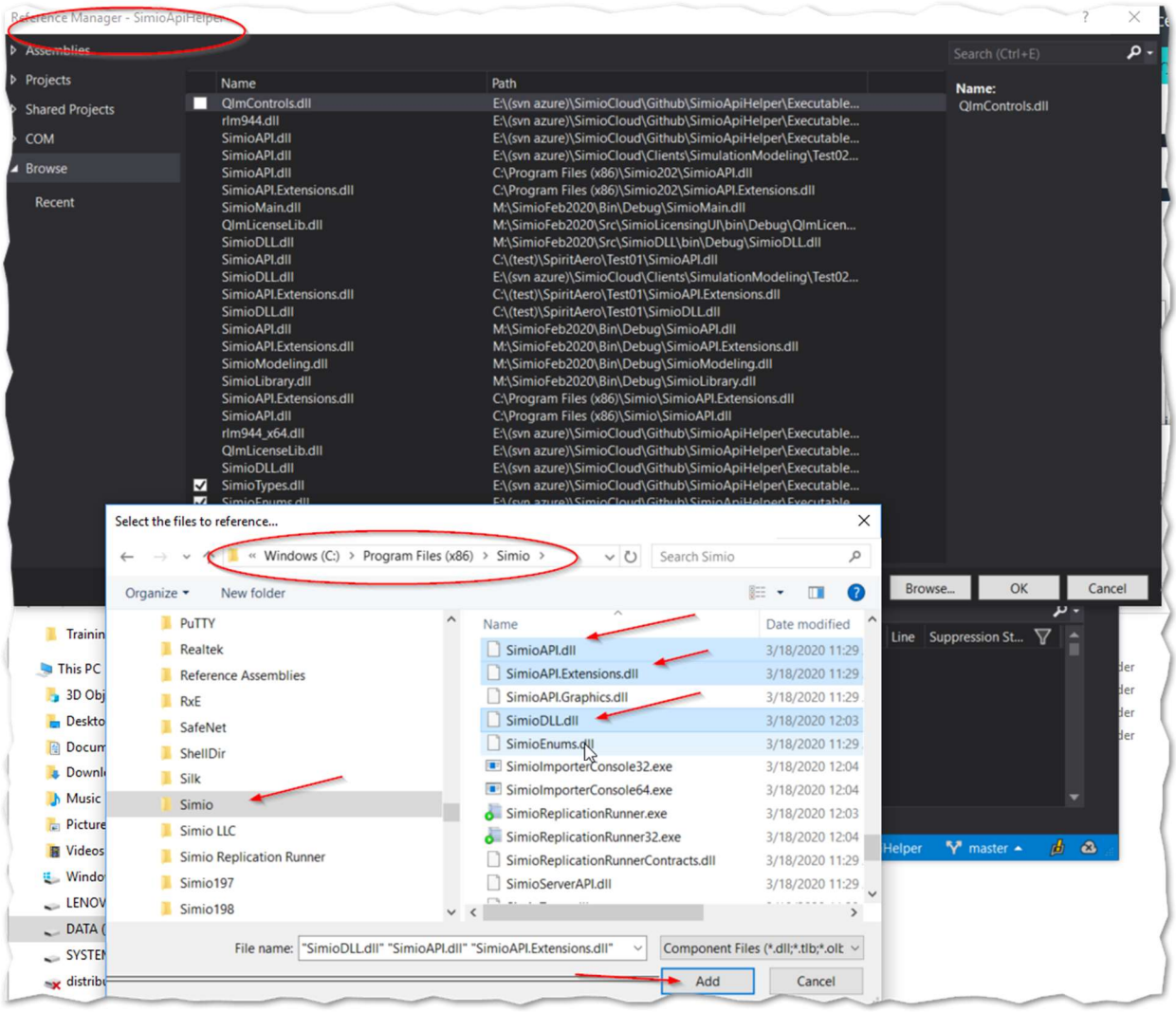
Simio

Forward Thinking



Next, right click on References and select “Add Reference…”, which brings up a dialog box.

At the lower right there is a button “Browse…” that will allow you to find your Simio DLLs:




You may have to do this same procedure for each project. It may seem a bit annoying, but it is a worthwhile exercise, and it is making sure that you are using the correct Simio libraries. Once done, you won’t have to repeat the exercise.

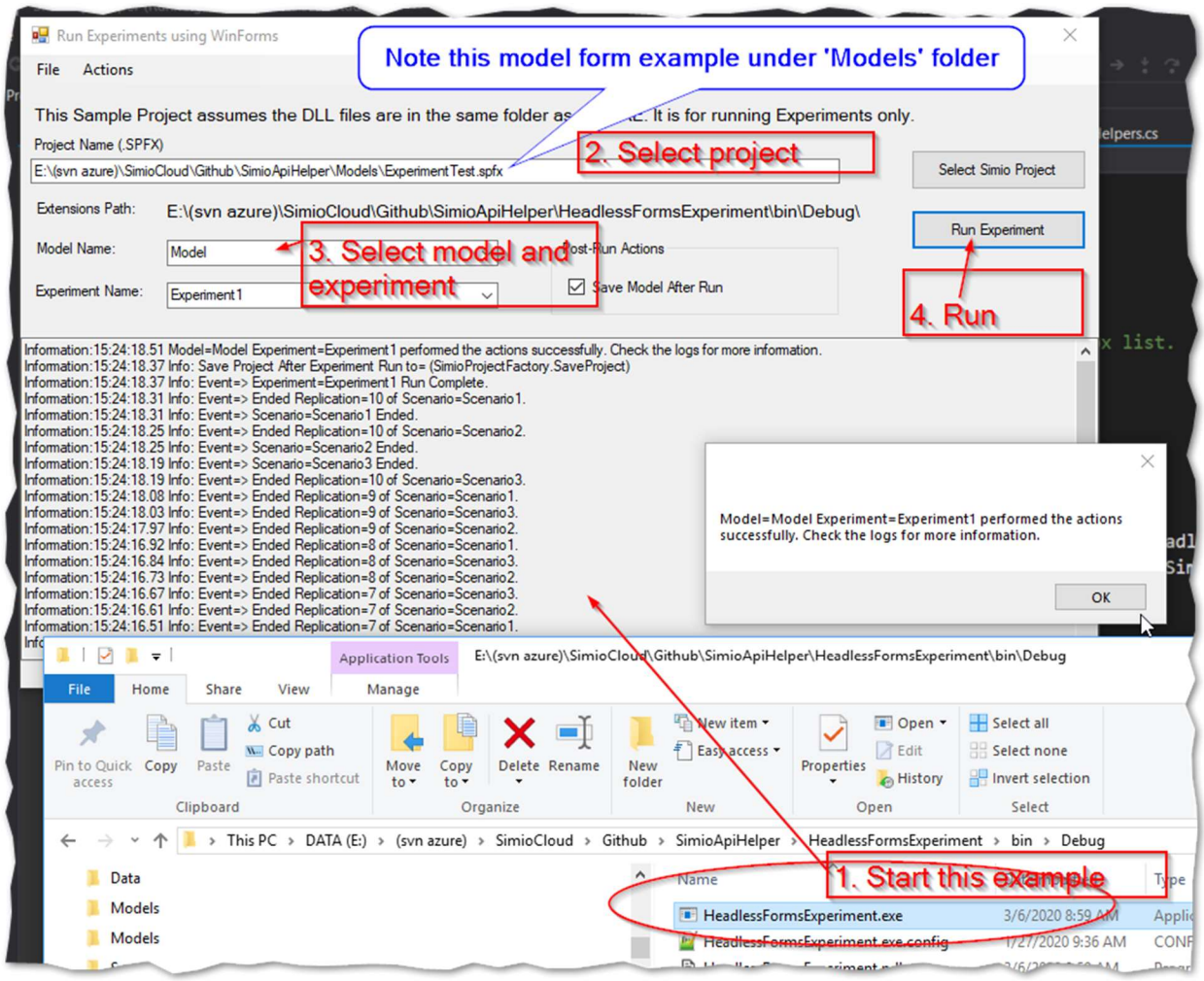
Simio
Forward Thinking

# Appendix: Headless Debugging

One of the hardest things to determine is what DLLs are required, and/or what the dependencies between the DLLs is.

There are two free tools that can help with this:

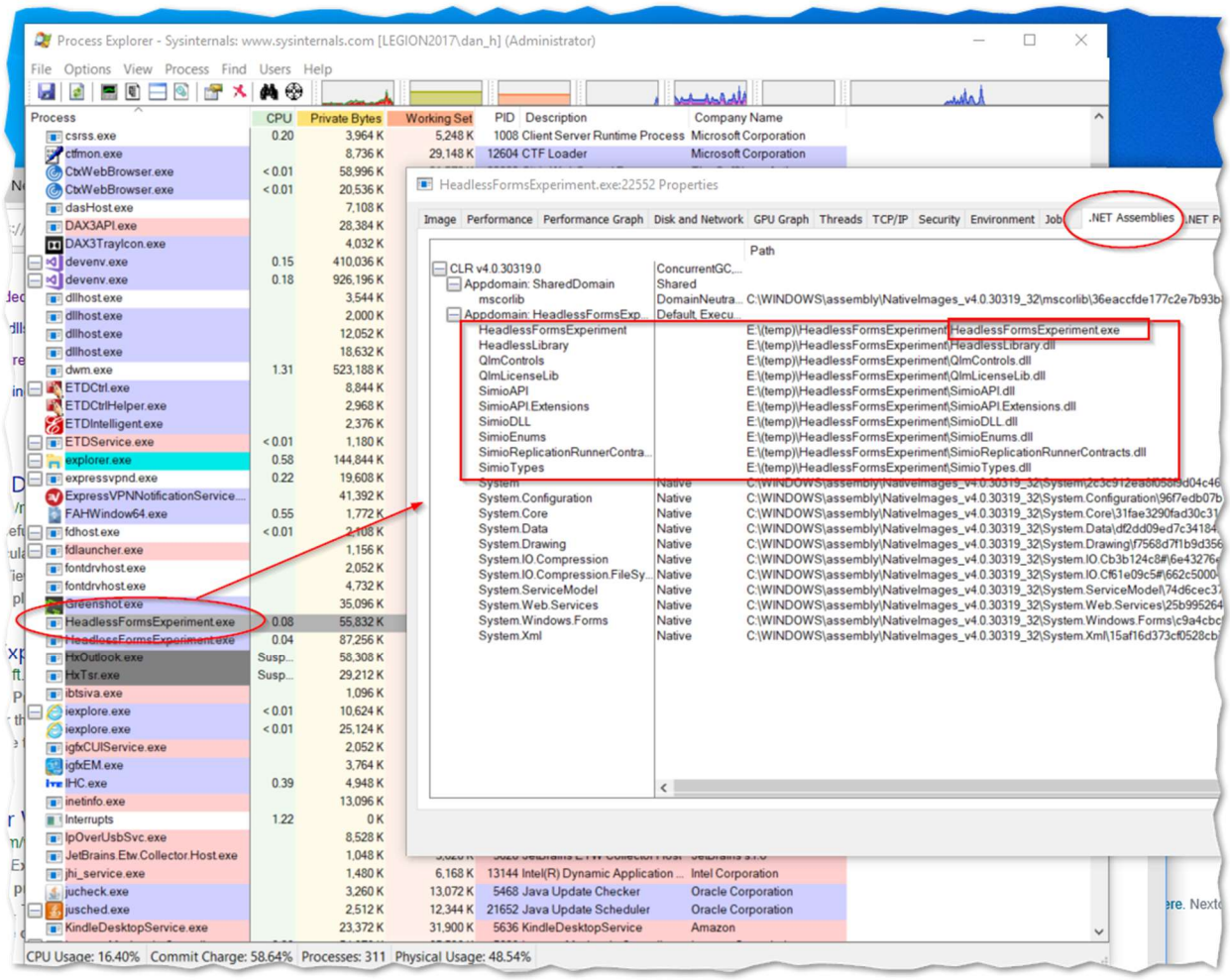
1. Process Explorer from SysInternals (Microsoft) to examine DLL dependencies
2. DotnetPeek from JetBrains to examine assemblies (such as DLLs)



Simio
Forward Thinking

Process Explorer can be used to examine a running program. This is incredibly useful because we can see what DLLs are employed regardless of when they were loaded. It can be downloaded for free from:

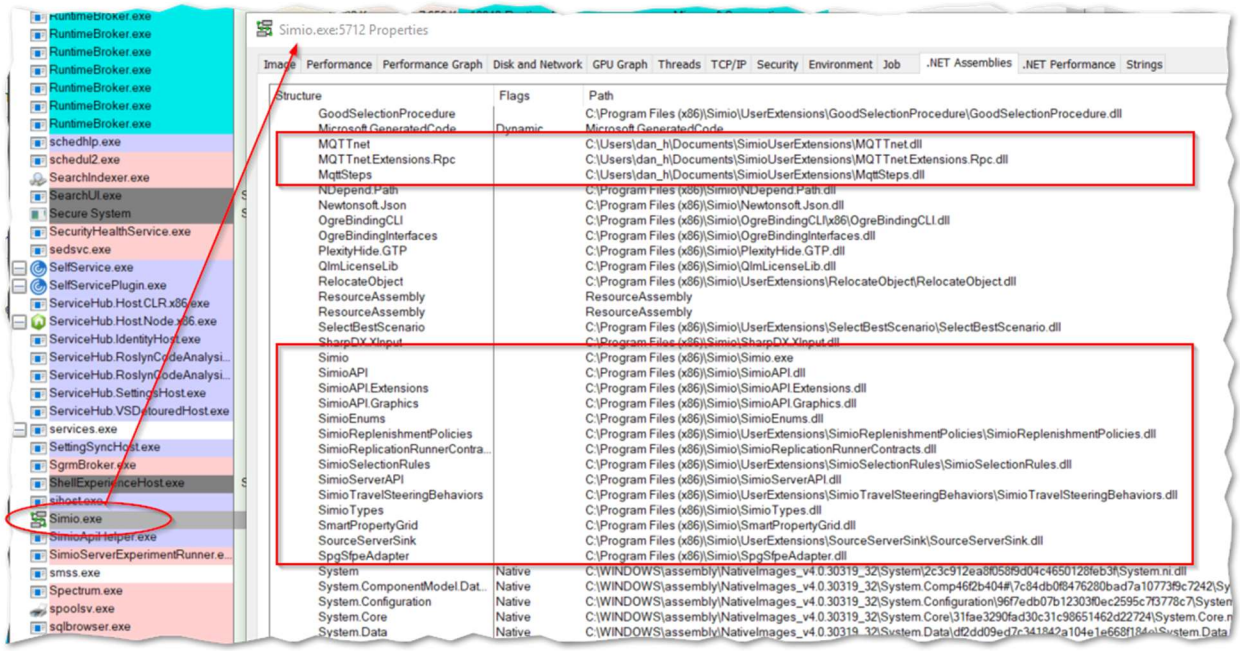
https://docs.microsoft.com/en-us/sysinternals/downloads/process-explorer




So, in the example above the included DLLs are shown.

Below is Process Explorer being applied against Simio with the same model being run.

Simio
Forward Thinking




And below is the result of one of the example projects “HeadlessFormsExperiment” which uses the call “System.Appdomain.CurrentDomain.BaseDirectory” to pick up the location of the HeadlessFormsExperiment.exe to locate all of the DLLs.


Simio
Forward Thinking

# Appendix – Simio Licensing (Server and Node-Locked)

This appendix describes some of the issues surrounding Simio licensing. When running in headless mode, Simio of course will still require a license, so it is important to make sure that your program is able to properly find its license. Machine based licensing (also called Node-Locked) is usually without problems, as the license information is simply on the machine. The difficulties usually arise with Server licensing, where the machine must locate the License Server on the network. The usual suspects of network location and firewalls then come into play.

## Server Licensing

The best documentation about Server licensing can be found in this document:

Here are some key points:

The licenses are stored on the License Server under ProgramData > Simio LLC > Simio Network Licensing.

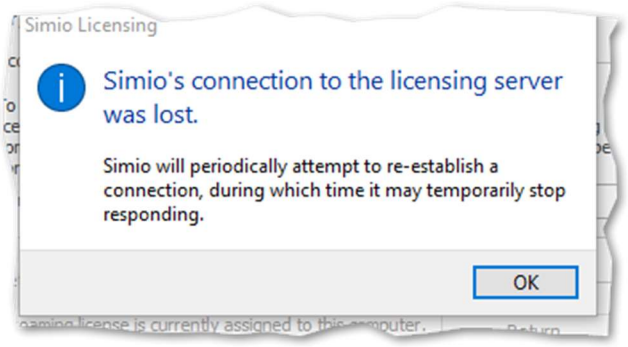
If there is a file !SimioConfig.lic, it hold configuration information about the “random” port used by the license simulator.

The files rlm.dlog and simio.dlog hold debugging information.

The files that have the extension “.LIC” hold the licenses.

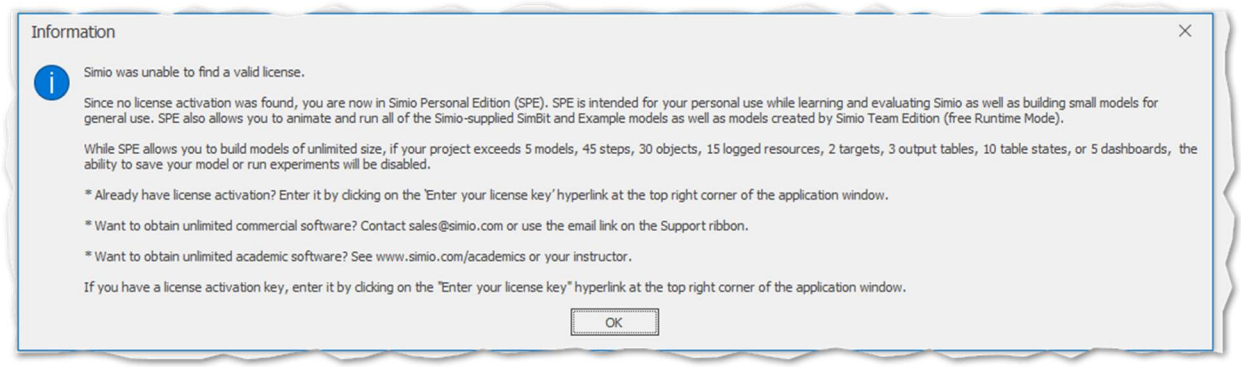
The Service on the License Server that runs licensing is named “RLM Simio”. It obviously must be running.

If it is stopped, the desktop Simio programs will raise a message box:




If the Server has no licenses, you get the message:

Simio
Forward Thinking




For more information, please see the document “Simio Network License Server.pdf” that accompanies this document.