# Simio API Note: Simio API Helper

Mar 2020

Contents

[Simio API Note: Simio API Helper 1](#_Toc34569319)

[Overview 2](#_Toc34569320)

[Using the API Helper. 3](#_Toc34569321)

[Tab DLL Helper 4](#_Toc34569322)

[Tab .Net Versions 5](#_Toc34569323)

[Tab: Find User Extensions 6](#_Toc34569324)

[Headless Workflow and Recommendations 7](#_Toc34569325)

[Tabs Headless Builder and Headless Run 10](#_Toc34569326)

[Headless Debugging 11](#_Toc34569327)

[Installing the Simio API Helper 15](#_Toc34569328)

[Appendix – Simio Licensing (Server and Node-Locked) 16](#_Toc34569329)

[Server Licensing 16](#_Toc34569330)

# 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:

# Using the API Helper.

The Simio API Tester is not part of the Simio product. Rather, it is simply a test tool that was built to help debug problems related to using the Simio API. How much benefit you derive will depend upon your programming knowledge, but anyone attempting to use an 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

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.

A screenshot of a computer screen

Description automatically generated

## 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.

A screenshot of a social media post

Description automatically generated

## 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.

A screenshot of a cell phone

Description automatically generated

## Headless Workflow and Recommendations

When you are running Simio from desktop you are running the Simio “Engine” with a bolted on Simio UI. This is the most popular 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 custom “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 it (via experiments or scheduling) storing the resulting data out to another database. 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 (called “assemblies” or DLLs). It is a bit confusing because Simio is very modularized and determining which DLLs to use can be a confusing task. It is further confused 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 data you are binding with.

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

A close up of a map

Description automatically generated

Let’s start with a recommendation: when you wish to have a headless configuration, it is wise to collect all the components you need into a dedicated folder, such as c:\SimioHeadlessApp\ ). 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, 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 these DLLs and put copies in a folder of your choice.

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.

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.

A close up of a map

Description automatically generated

### The Code for a Headless Operation

The actual code is quite simple and generally of the form

1. Look for a reason to run, such as new data arriving
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

### Tabs Headless Builder and Headless Run

A screenshot of a social media post

Description automatically generated

### 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)

A screenshot of a computer

Description automatically generated

ProcessExplorer 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>

A screenshot of a computer

Description automatically generated

A

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

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

A screenshot of a computer

Description automatically generated

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.

Tab Logs and Settings

# Installing the Simio API Helper

The helper is just an EXE (SimioApiHelper.exe) and is built with .NET 4.5.2.

There are no other dependencies (e.g. DLLs) beyond .NET.

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

## 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:

A screenshot of a cell phone

Description automatically generated

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

A screenshot of a social media post

Description automatically generated