Simio
Forward Thinking

Simio API Note:

Creation: June2018 (Dhouck)

Revisions: November 2019 (Dhouck) – Added to Github

Contents

Simio API Note: 1

Overview 2

Architecture 3

Some Background Information on the Simio Engine 5

Building A Headless Executable 6

Creating a Service From the Executable 10

Simio
Forward Thinking

Overview

This API Note describes how Simio can be configured to do the following:

1. Run a Schedule in a “headless” or unattended mode to create a Model Plan
2. Export the results from the schedule.

Although Simio is usually used with the desktop User Interface (UI) it can be configured to run unattended or “headless”.

This project will demonstrate a configuration where a model is set up to be run as a Windows service, and be triggered whenever a file is dropped into a give folder.

This Note describes some complex programming topics. It assumes that the reader is familiar with C# and .NET technologies such as locking and DLL dependencies.

Simio
Forward Thinking

Architecture

The diagram below illustrates the overall architecture of this API example:

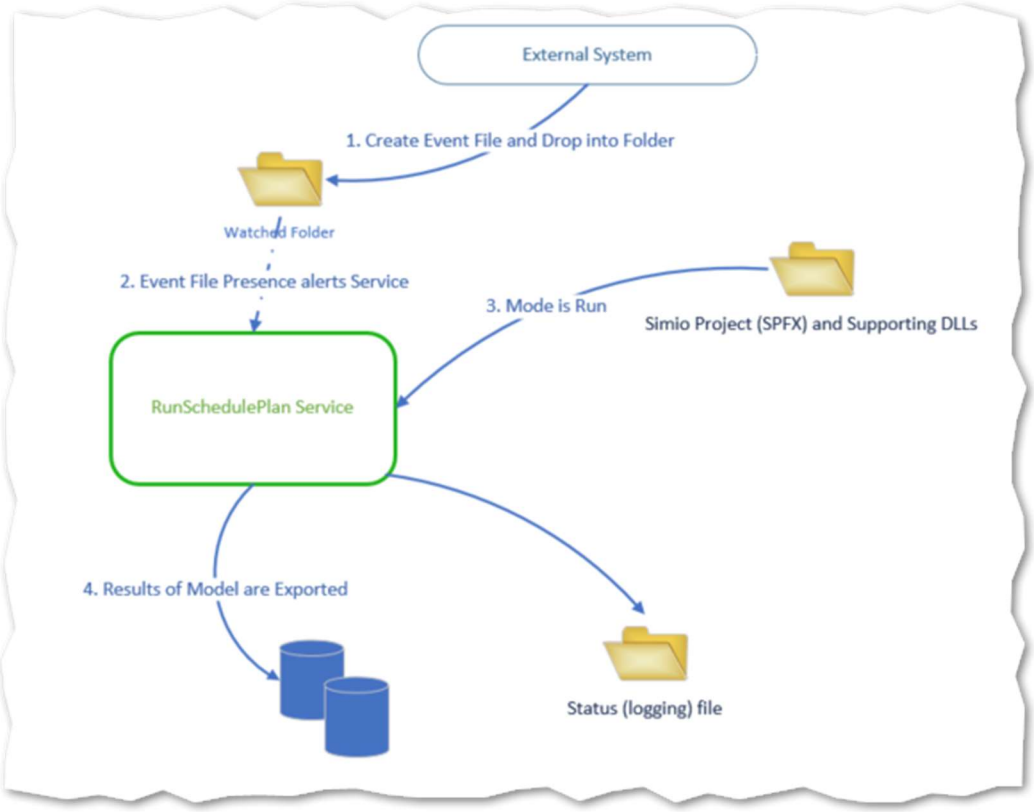


Figure 1- RunSimioSchedule Architecture

RunSimioSchedule is a windows process that runs as a service and waits for an event file to be dropped into a specified folder.

As the program begins the method Main is called which creates a RunContext which will exist throughout the life of the service. During the RunContext constructor, the following occurs:

- The API SimioProjectFactory.SetExtensionsPath is called to indicate to the model where all the support DLLs can be found.
- Some setup occurs, such as making sure of the existence of the Event file folder.
- The Simio project file (e.g. .SPFX) is located and loaded.
- Calls OnStart which creates a system FileWatcher which monitors the Events folder and has events for whenever an event file is created or changed within the folder. It also starts a timer that polls for the event file.

Then the program goes into an infinite loop, leaving the Event file event checking to do all the work.

Whenever an Event file is found, the method CheckAndRun is called, which subsequently calls RunScheduleExportResultsAndSaveProject, which is the central logic to this example.

RunScheduleExportResultsAndSaveProject does the following.

- Converts the Event File - which is a CSV (Comma Separated Value) file to a DataTable and then deletes the Event File.
- Gets the model from the already loaded Simio project.
- If the DataTable exists, then it is assumed to be downtime data and is Imported into the model's Resource Table the method ImportDowntime. If the model doesn't have a table called "Resources", then the data is simply ignored.
- If configured, the Model is saved prior to the run.
- If configured, the model's Plan is run.
- If configured, risk analysis is run on the model.
- If configured, the plan's schedule is exported.
- If configured, the model is saved

Simio
Forward Thinking

Some Background Information on the Simio Engine

The Simio engine (or simply as the Engine) is the logic that implements the simulation and planning logic of Simio. When used as a desktop application, it is paired with a UI, but in the “headless” mode it isn’t. Only the DLL containing the engine (SimioDLL.DLL) along with support files for API and some file operations will be used.

When running within the UI specific folders are searched by Simio when looking for DLLs, such as User Extensions and other add-ins.

When running headless this is not done, and instead all DLLs that are used must be in the location specified by the call SetExtensionsPath of SimioProjectFactory.

This can be a tricky problem, as often DLLs depend on other DLLs, which depend on even more and so on. There are a few free tools that might help you solve this puzzle:

One is DotPeek, which is made by JetBrains. With this tool you can inspect a DLL for its dependencies.

Another is Process Explorer, which part of Microsoft’s SysInternal toolset. When run in Administrator mode this tool permits you to examine a running process (such as Simio) and determine what DLLs are loaded.

In this document are included the instruction for loading your headless executable as a Windows Service..

Simio
Forward Thinking

Building A Headless Executable

The overall structure of the program is simple: a path to the extensions is set, and then a series of actions on the model, such as running the model plan or running risk analysis are issued.

The difficulty most users have – as was mentioned before – is determining the correct DLLs to include. This will be covered later in a separate section.

Separate but closely allied with the calling of the model is the need for a mechanism to initiate the model. This example employs a file-drop mechanism. When a file is dropped in a special folder, a method is triggered by the System FileWatcher. This method (called CheckAndRun) runs the method RunScheduleExportResultsAndSaveProject, which does most of the work.

Note: because of a well-known FileWatcher deficiency (under rare circumstances it won't detect a file event) there is also a timer that checks every so often for a file and runs the same model processing method.

There is also a file that is logs information about the running of the model. This is specified in the configuration file.

The utility methods (such as those that are used to read and write information to the model are included in a utility class called HeadlessHelpers.

Configuration settings are included as Application Settings, which – during the build of this application are output into a configuration file (RunSImioSchedule.exe.config) that is placed in the same folder as the executable (RunSimioSchedule.exe).

	Name	Type	Scope	Value
▶	EventFile	string	User	C:\\(test)\\RunSimioScheduleTest\\Event.csv
	StatusFile	string	User	C:\\(test)\\RunSimioScheduleTest\\RunStatus.txt
	SaveProject	bool	User	True
	ClearStatusBeforeEachRun	bool	User	False
	DeleteStatusBeforeEachRun	bool	User	False
	SimioProjectFile	string	User	C:\\(test)\\RunSimioScheduleTest\\Model v92.0.spfx
	ExportScheduleFile	string	User	C:\\(test)\\RunSimioScheduleTest\\ExportSchedule.xml
	ModelName	string	User	Model
	RunPlan	bool	User	True
	RunRiskAnalysis	bool	User	False
	ExportSchedule	bool	User	False
*				

When the program is built from Visual Studio the results are placed (as always) in a folder such as Source > RunSimioSchedule > bin > release, such as:

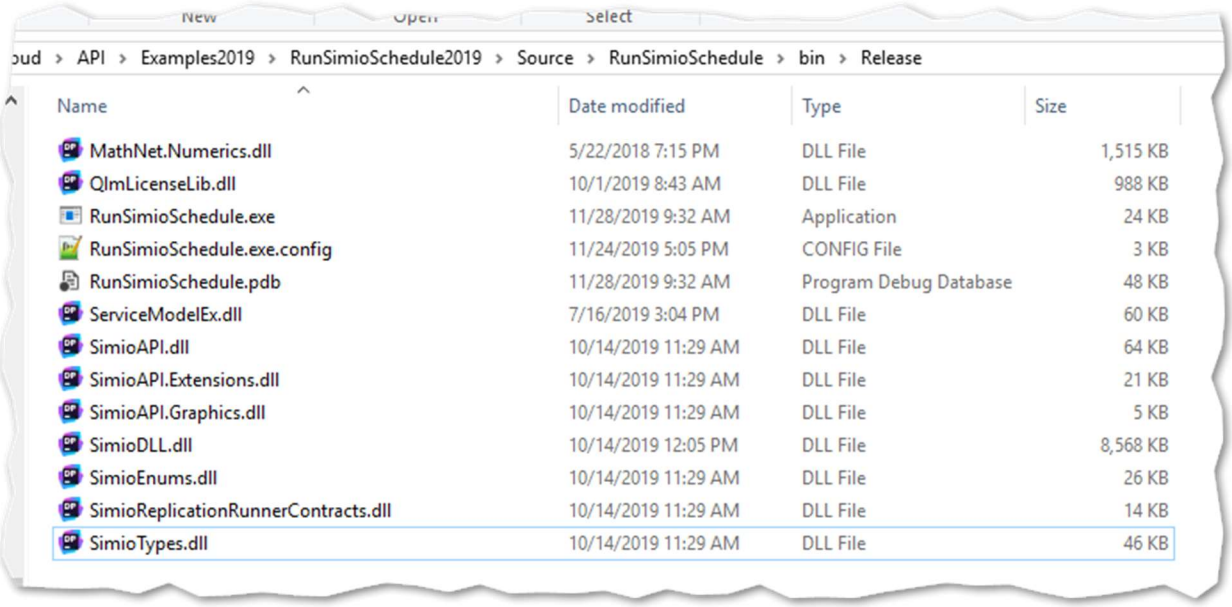


Figure 2 - Assemblies From VIsual Studio - Very Likely Incomplete!

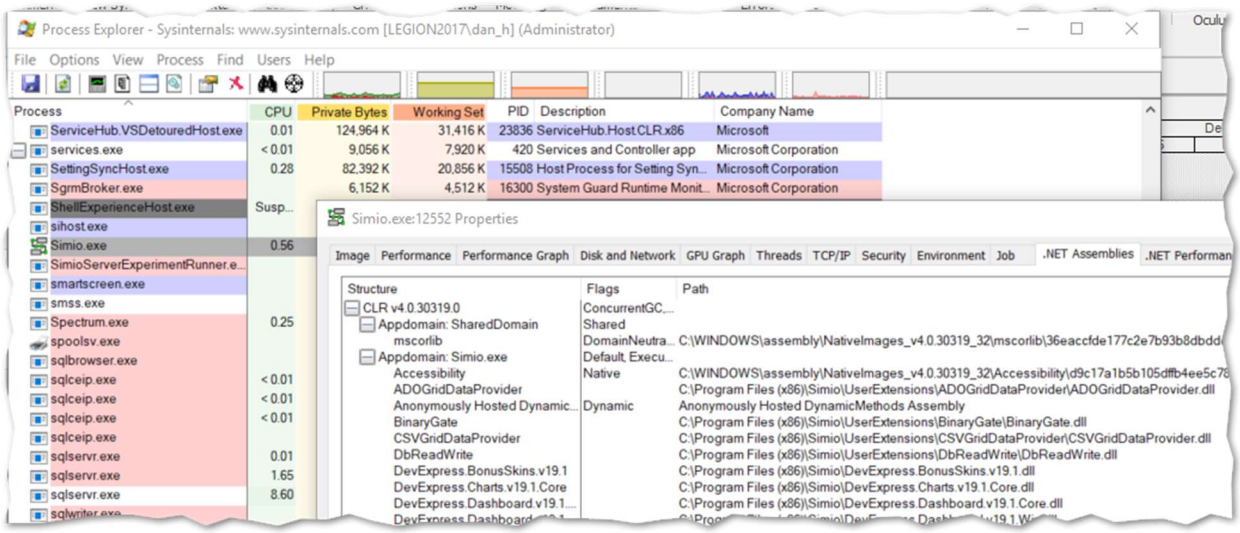
Now, you might think that all the DLLs you need would be found here. But this is likely wrong, since these are only the assemblies that Visual Studio could find through its process of compiling and linking and explicit references. The RunSchedulePlan program is going to – at run-time - launch a Simio Model, which may include many other DLLs (such as licensing files, user extensions, etc.)

Simio
Forward Thinking

So, what to?

A good starting point is to run Simio with our model and then run Microsoft's ProcessExplorer (as Administrator) to examine what DLL's are being used.

After starting Simio desktop and doing a RunPlan, Process Explorer is started. Look for Simio.exe and double-click to bring up its DLL view:



Looking through this we can see a lot of DevExpress DLLs, but many are DevExpress UI DLLs. There is a reference to DevExpress.Office.v19.1.Core, which is often included when Excel operations are employed.

Of great interest to us are the ones within \SimioUserExtensions:

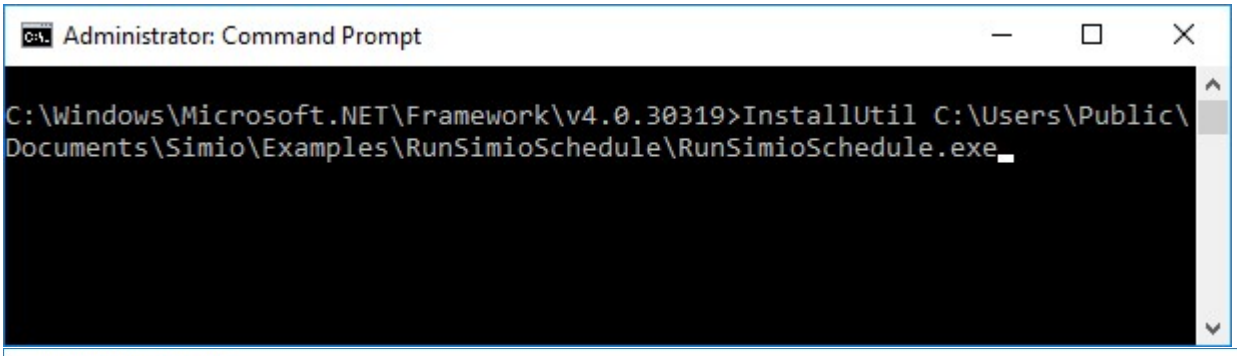
- ExcelGridDataProvider
- ExcelReadWrite
- GoodSelectionProcedure
- SimioRelocateObject
- SelectBestScenario
- SimioReplenishmentPolicies
- SimioSelectionRules
- SimioTravelSteeringBehaviors
- (no) SourceServerSink
- (no) WonderwareMES

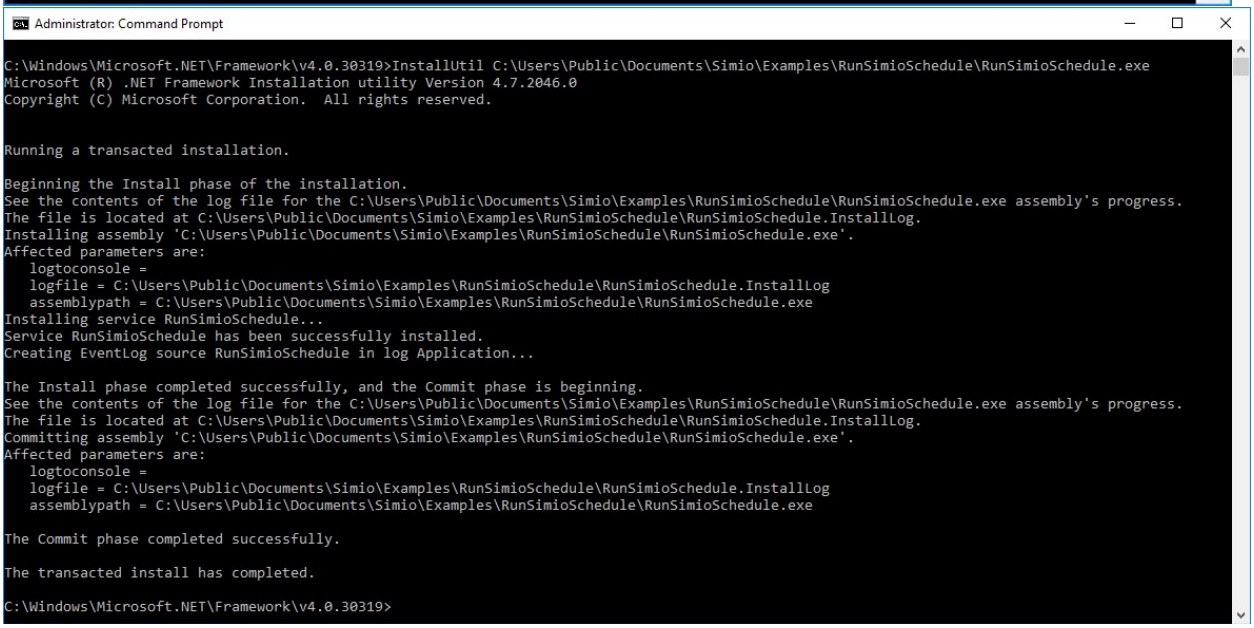
- XMLGridDataProvider

Simio
Forward Thinking

Creating a Service From the Executable

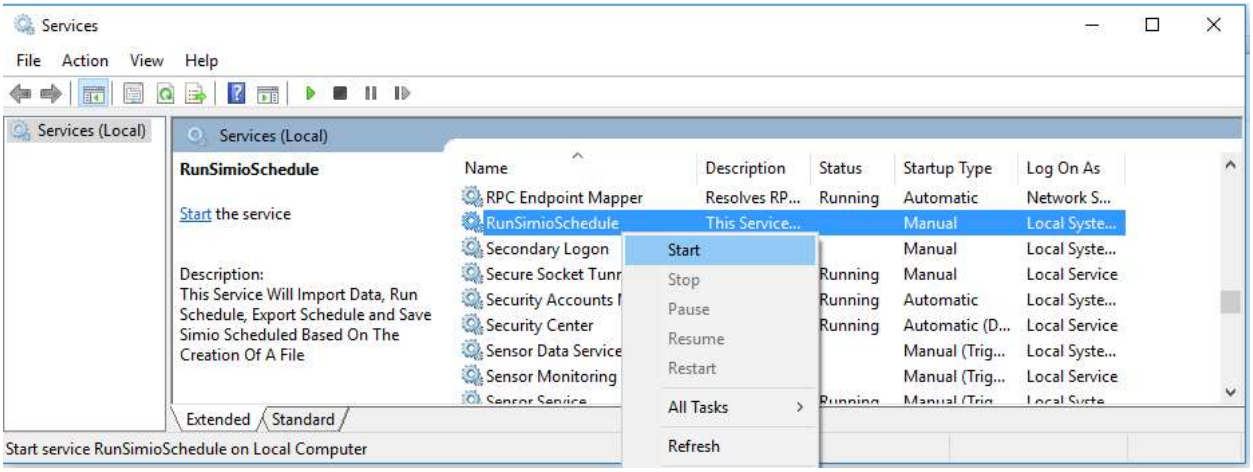
1) Open a command prompt as Administrator
2) Navigate to your .NET installation folder. For example: "C:\Windows\Microsoft.NET\Framework\v4.0.30319> "
3) Enter "InstallUtil followed by the path to your executable. For example: C:\Users\Public\Documents\Simio\Examples\RunSimioSchedule\ RunSimioSchedule.exe". This will install the RunSimioSchedule as a windows service.

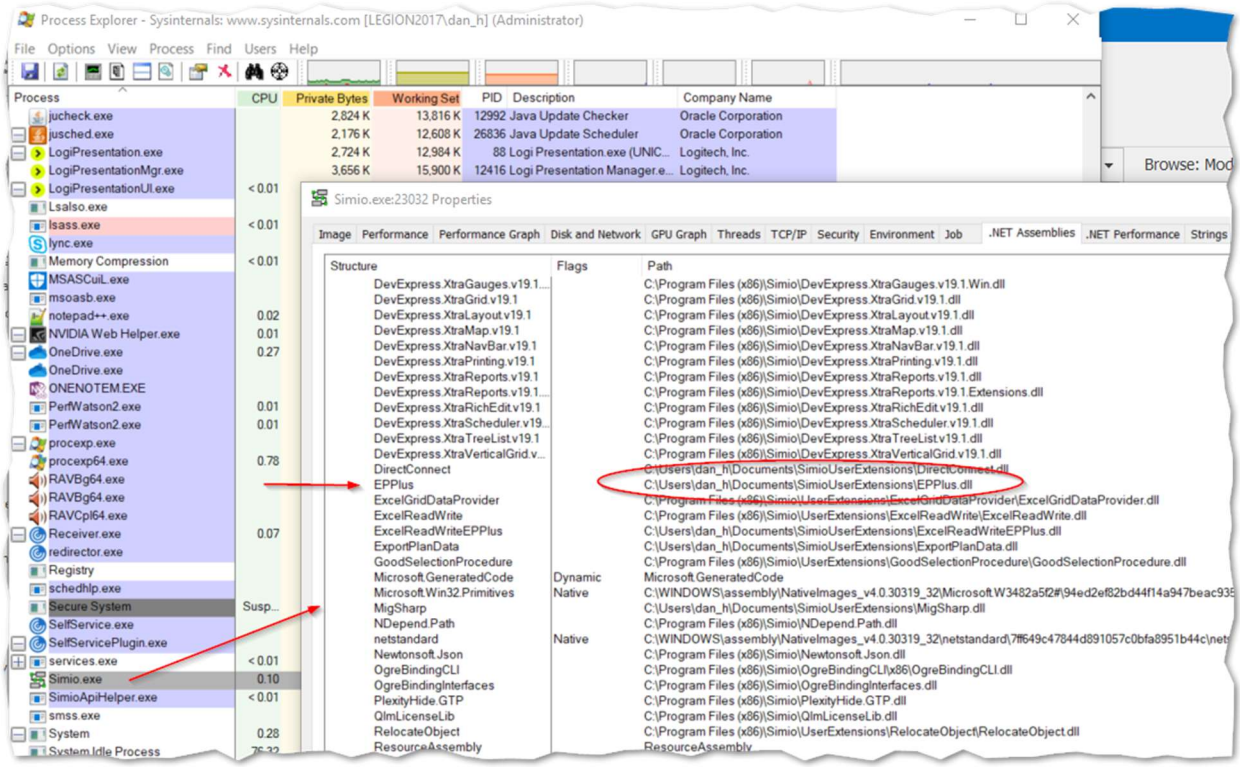




4) Next, start the service from Services.

Simio
Forward Thinking



Simio

Forward Thinking