SRTI - SIMPLE RUN-TIME INTERFACE

v2 GUI Documentation

V0.69.01 – 2019-09-25

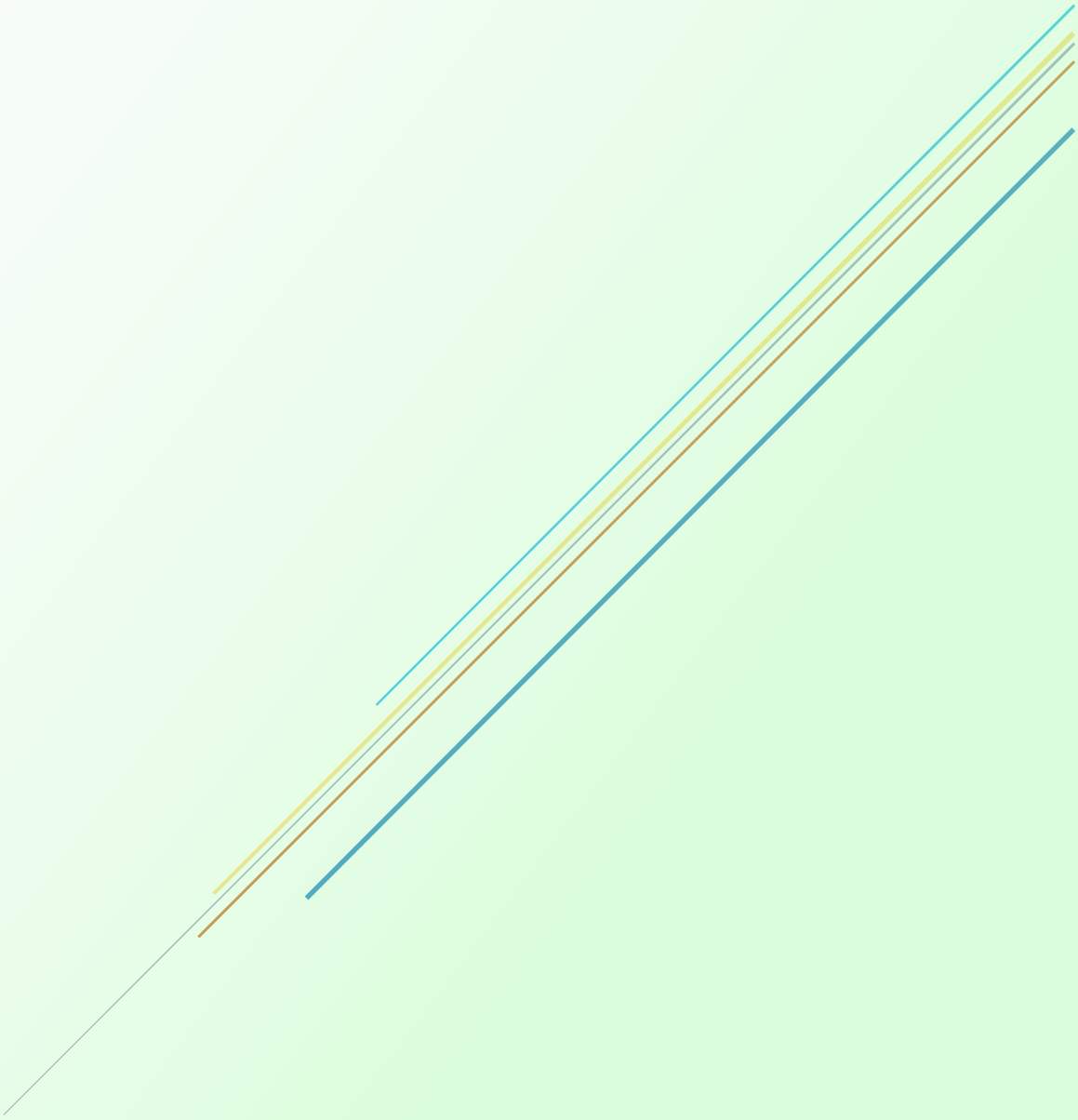


Table of Contents:

Chapter 0 – Release Notes

i. Summary

This is an introduction to the SRTI v2 GUI, an optional complimentary software to use with the SRTI v2 system. This free, open-source application was written in HTML, CSS and JavaScript, compiled with Electron. This is not to be confused with the "Server GUI" that is written in Java's Swing library, already included inside the SRTI .jar file.

For compatibility, the SRTI v2 GUI (**v0.68.1**) should be used with SRTI **v2.20.02** (both RTI Server and Wrapper), or else the most recent up to that version.

SRTI (Simple Real-Time Interface) is a portable software solution to allow data transfer between different programs, be they in different languages or on different computer systems, be they local or connected through a network. This was built with the intention of being easy to use and maintain, primarily for scientific simulations to cooperate with each other in real time but can also be utilized in related IoT (Internet of Things) projects.

The project began in 2017 and is still under continuous development to add functions and improve efficiency. SRTI is free and open-source and is funded in part by the University of Michigan.

The full public source code, pre-compiled libraries, documentation, and other information can be found at https://github.com/hlynka-a/SRTI .

ii. Release Updates

2019-09-12

- This covers SRTI v2 GUI v0.68.1, which includes a compiled desktop GUI for the Windows operating system that can output configuration files for a SRTI v2 Wrapper, and launch a series of simulators from within the GUI itself. This is considered to be an alpha version of the app, which may contain bugs, and may be subject to changes by the time the full version is released.

Chapter 1 – Introduction

i. What is the SRTI v2 GUI?

SRTI (Simple Real-Time Interface) is a free, open-source and portable software solution that allows external software simulations to connect with each other and share information in real-time.

SRTI v1.00.00 is purely a data-transmission system. It uses a single RTI Server as the shared access point for individual simulator programs to join. Each simulator needs to locally reference the provided RTI Lib API to make a connection, which abstracts most of the low-level details of the connection. Sockets are used for the communication channel between RTI Server and RTI Lib. Messages sent through the system are transmitted in 'string' format, for better interoperability between languages, compared to pure byte code. To assist with this, JSON is used as the standard message format (although any string-representation can be used for the content from each simulator).

SRTI v2.00.00 is a more advanced and specialized update of v.1.00.00. Building upon the original data-transmission protocol, it adds new functionality based on a different goal: to explicitly support artificial simulation systems. In addition to the RTI Server and RTI Lib API, it adds an RTI Manager (coupled with RTI Server) and RTI Wrapper. Instead of having simulators control themselves and how they parse information from the SRTI, the RTI Wrapper takes over much of the responsibility, allowing easier design of larger simulation systems without explicit recoding.

The SRTI v2 GUI is a helper application meant to work with SRTI v2.00.00. Normally, v2.00.00 requires a configuration file to be written for each simulator, written in JSON format according to a set definition. After this, the user would have to launch every simulator individually to connect to the RTI Server. The SRTI v2 GUI makes these steps easier: the user gets an **interactive graphic interface** to describe their simulator system (and how different simulators correspond to each other), and it can **output the configuration files based on the user's design**. Buttons inside the GUI can also **launch the RTI Server and simulators**, rather than having the user to it manually one at a time.

> **DISCLAIMER**: It is **strongly recommended** that the user read through the documentation for SRTI_v2_00_00, to understand how the SRTI, the Wrapper, and the configuration variables work, before attempting to use the SRTI v2 GUI, and before reading this document.

ii. Downloading and Launching

The SRTI v2 GUI can be downloaded from the same public location as the rest of the SRTI files, on GitHub. The root branch can be found at https://github.com/hlynka-a/SRTI , and the compiled GUI can be found at https://github.com/hlynka-a/SRTI/tree/master/SRTI_GUI/compiled_example/gui .

Compiled with Electron and "electron-builder," there are 2 versions of the SRTI v2 GUI that can be downloaded. The first is a single standalone .exe ("SRTI-v2-00-00-Manager-GUI 0.68.1.exe"), above the "win-unpacked" folder. The second is the entirety of the "win-unpacked" folder, including another .exe ("SRTI-v2-00-00-Manager-GUI.exe"). They are both functionally identical; the standalone .exe is a packed version of the other, easier to download and copy between computers. Consequently, opening the standalone .exe may take several seconds (it must unpack itself), whereas opening the "win-unpacked" version is quicker. Both versions do not need to be formally installed, and can be transferred between systems or folder locations.

Both versions include a version of the RTI Server ("SRTI_v2_20_02.jar") that it references. A function is in place to allow the user to reference a different version (currently not functional) within the GUI.

As of 2019-09-12, only a Windows version of the app is compiled. The source code can be downloaded on Mac and Linux machines to compile a version for other operating systems (Electron software framework required to compile: see http://electronjs.org). For basic testing, the GUI can also be opened in a web-browser on a local machine, but access to a local file system from a browser is restricted, preventing saving/opening projects or launching a system.

To open, double-click on the .exe.

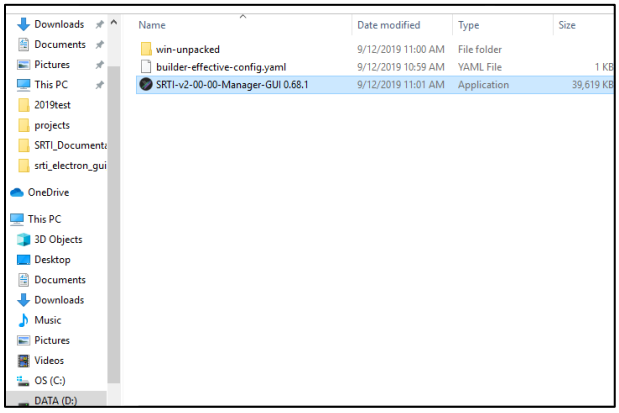


Figure 1 - Location of packed standalone GUI .exe

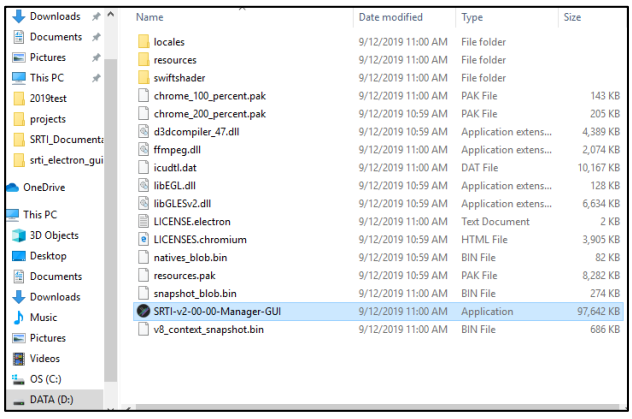


Figure 2 - Location of unpacked GUI .exe

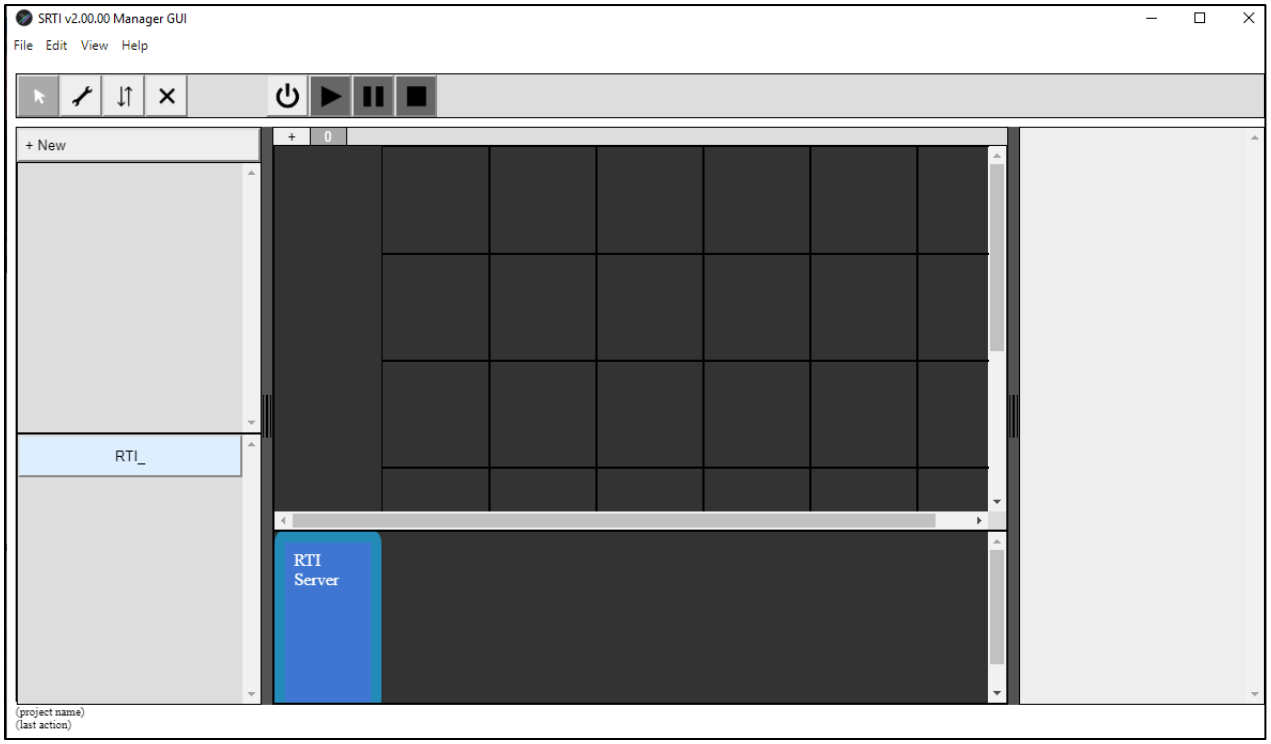


Figure 3 - SRTI v2 GUI with empty project.

iii. Preparing a File System

The SRTI v2 GUI can be downloaded and saved in any file directory on your Windows machine without explicitly installing it, making it easily portable. However, when saving a SRTI Project, the GUI will produce multiple files, making organization difficult. This is further compounded when the user starts referencing multiple simulators on their computer.

For now, it is recommended that the user prepare a simple file directory to look like the following. This is not required, but recommended.

- **/Gui/**
  - Store the packed or unpacked SRTI v2 GUI files.
- **/Projects/**
  - Store individual folders for each project you make using the SRTI v2 GUI. For example…
  - **/Project_01/**
  - **/Project_02/**
  - **/Project_03/**
  - **…**

- **/Server/**
    - If required, store the SRTI v2.00.00 Server type you want to reference. A version is already included with the GUI.
- **/Sims/**
    - Store individual folders for each simulator executable (and a corresponding Wrapper) you plan to reference in the GUI. For example…
    - **/Sim_A/**
        - SRTI_JavaWrapper_v2_20_02.jar
        - SimA.jar
    - **/Sim_B/**
        - SRTI_NetLogoWrapper_v2_20_02.jar
        - SimB.nlogo
    - **/Sim_C/**
        - SRTI_Wrapper.m
        - SimC.m
    - **…**

Assuming this is the file structure, what would be found in the /Project_01/ after saving it? You'll see files with extensions as described below. Despite the extensions appearing proprietary, each file represents ASCII-text data (typically representing a JSON object) that can be read in most text editors.

- .project
    - Represents a single large JSON object that defines the full project. 1 of these files exists per project.
- .simdef
    - Optional output file the describes a single simulator referenced in the .project file. It isn't necessary to re-open the project, but can be imported into a new project to prevent re-defining a simulator reference from scratch.
- .mesdef
    - Optional output file the describes a single message referenced in the .project file. It isn't necessary to re-open the project, but can be imported into a new project to prevent re-defining a message reference from scratch.
- executeCommands.txt
    - If for any reason the GUI is unable to execute the system by itself, this file lists commands that could be run in the Windows Command Prompt to launch the system separately from the GUI. Each line would have to be run manually, or through a custom script from the user.

iv. GUI Limitations

There are a few features that the SRTI v2.00.00 system does support, but are difficult to work into the GUI.

SRTI v2.00.00 assumes that all connected simulators use a “Wrapper” class to handle the connection between a Simulator and the RTI Server. This is not strictly required, but functionality may not proceed as expected if a user’s simulator doesn’t send expected messages. Similarly, the v2 GUI can execute simulators directly instead of a Wrapper, but behavior may not be as expected: the purpose of the GUI is to output Wrapper configuration files that describe timing, message subscribe and publish parameters, etc. If a Wrapper isn’t used, then definitions described visually in the GUI won’t apply to the real execution.

SRTI v1.00.00 would not be able to utilize the visual definitions of the v2 GUI (timing, publish and subscribe, etc.). v1.00.00 was intended as a bare-bones version of the SRTI that would allow and require the highest level of customization by a user programmer, and doesn’t support the well-defined structure of SRTI v2.00.00 or the v2 GUI.

SRTI v2.00.00 and v1.00.00 support the ability to connect simulators through a network connection (it is recommended that the RTI Server, and if possible, the simulators, run on a static IP to maintain a consistent connection). However, the v2 GUI would be unable to connect to external computers to run a command. In theory, a remote connection system could be implemented inside the v2 GUI, similar to SSH client software (needing to input the connection parameters), but this is not supported at this time. Alternatively, a user can execute external simulators by hand when launching the RTI Server from the v2 GUI, but a representation of it would not be present in the GUI’s visuals.

v. GUI Layout and Functions

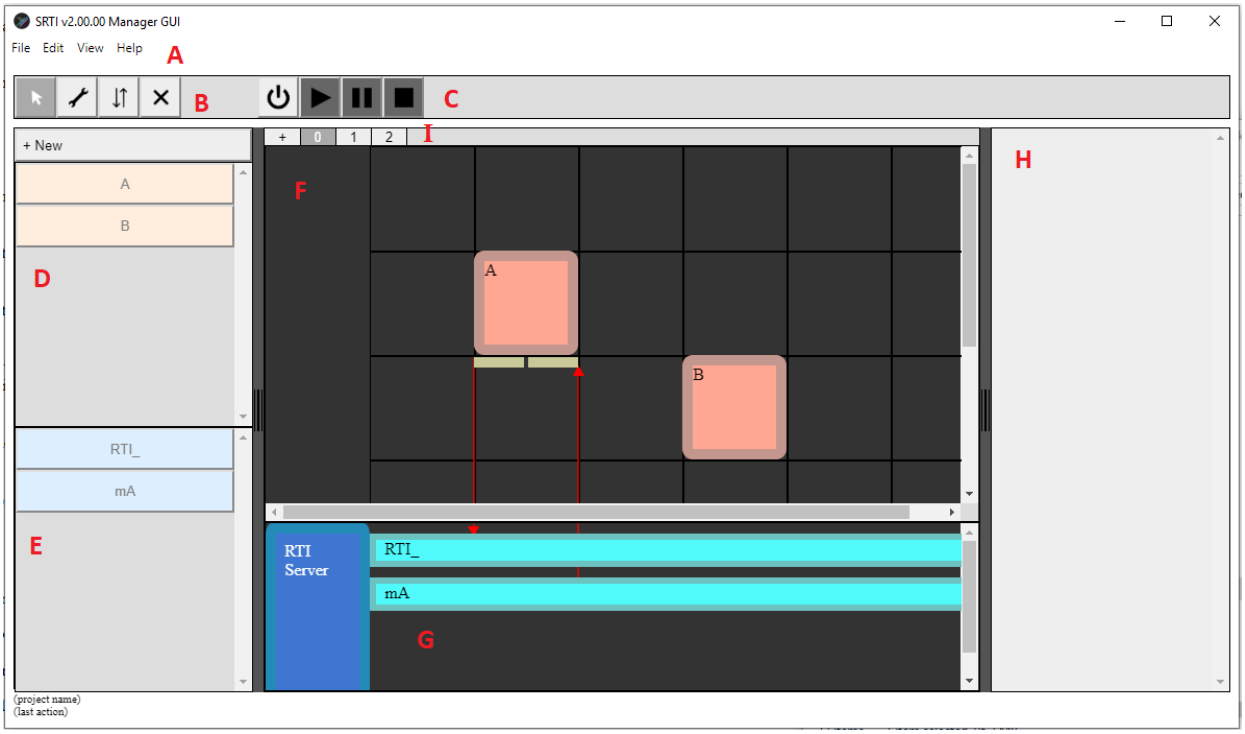


Figure 4 - Map of SRTI v2 GUI Layout

- **A: Menu Bar**
  - Access basic functions like "Start new project," "Open existing project," "Save/SaveAs", "Export Configuration Files (for Wrapper)," "Undo/Redo," and more. (some features may not be completed)
- **B: Action Toggle Buttons**
  - Allows user to change mode of action. Modes include (from left to right) "Select," "Configure," "Publish/Subscribe," and "Delete." Actions apply to all objects on the Canvas and Object List.
- **C: Execute Buttons**
  - Allows user to Start RTI Server and corresponding simulators, Play (begin or resume) the system, Pause the system, or Stop (shut down) the active system.
- **D: Simulator Object List**
  - The user can define new simulators here. When added to the Object List, it can be added to the Simulator Canvas.
- **E: Message Object List**
  - The user can define new messages here. "RTI_" is a default message to allow subscribing to proprietary messages
- **F: Simulator Canvas**
  - From the Simulator Object List, the user can add and drag simulator objects onto the canvas. The vertical position determines "order" of execution when running.

- **G: Message Canvas**
    - From the Message Object List, the user can add message objects onto the canvas.
- **H: Object Inspector**
    - When “Configure” Toggle is on, user can click on objects from the Object List or Canvas and specify/edit details about them here.
- **I: Stage Bar**
    - For complex systems that require multiple distinct stages of execution, the user can add new stages and change between them to show a new Simulator Canvas. Transitioning between them can be specified in the Object Inspector.

Chapter 2 – Grocery Store Example

The following section introduces a simple example to show how the SRTI v2 GUI can be used.

i. Definition of the Grocery Store

Let's describe the simulation system we are trying to make.

Assume a scenario where we have a Grocery Store "G." It keeps stock of apples, blueberries and carrots.

In addition to G, we have customers "C-A," "C-B" and "C-C." A comes in every day to buy apples. B comes in every 2 days to buy blueberries. C comes in every 3 days to buy carrots. G makes money each time a purchase is made, until G's stock of food runs out.

This represents a simple simulation system. G, C-A, C-B and C-C are the 4 simulators. C-A, C-B and C-C are reliant on G to purchase food if available, and G is reliant on them to earn money.

We could make this system more complicated in a few ways. We could introduce suppliers for G to contact to deliver more stock. Those suppliers could be dependent on individual farms. The customers could have a source of money dependent on their job or other income. For now, we'll assume these are not factors in this simple example.

Therefore, our simulation system is fairly straightforward. It only requires 1 stage. To prevent it from running forever, we'll add 2 end conditions: if G runs out of all of its stock, or if any one of the customers dies from running out of food.

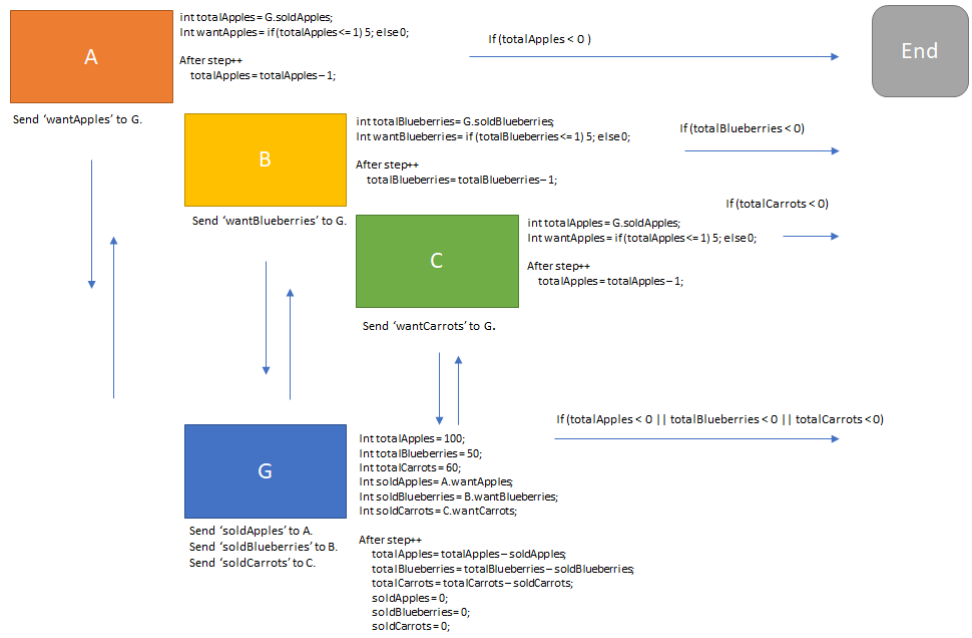


Figure 5- Design of Grocery Store Example

ii. Definition of simulators (Java, Matlab, NetLogo)

While you can define your own code in your own language of choice, using this example as a basic template, you may prefer to use the exact simulators directly to reproduce the intended result. For that reason, the simulators A, B, C and G are written below in Java, Matlab and NetLogo (the supported languages for which Wrappers have been prepared at this time). Compiled versions of this code will be used for the proceeding subsections.

For each simulator, you can use the language-version of your choice interchangeably with the SRTI.

Below are the simulators written in the Java language:

A (A.java)

```
public class A {

	public int totalApples = 4;
	public int wantApples = 0;

	public static void main(String [] args){
		// below is some sample code for if the user wanted to run this without the SRTI
		A thisClass = new A();
		for (int step = 0; step < 100; step++){
			if (step % 1 == 0){
				thisClass.UpdateNextStep();
			}
		}
	}

	public void Initialize(){
		totalApples = 4;
		wantApples = 0;
	}

	public void UpdateNextStep(){
		if (totalApples <= 1){
			wantApples = 5;
		} else {
			wantApples = 0;
		}
		totalApples--;
	}
}
```

B (B.java)

```
public class B {

	public int totalBlueberries = 4;
	public int wantBlueberries = 0;

	public static void main(String [] args){
		// below is some sample code for if the user wanted to run this without the SRTI
		B thisClass = new B();
		for (int step = 0; step < 100; step++){
			if (step % 2 == 0){
				thisClass.UpdateNextStep();
			}
		}
	}

	public void Initialize(){
		totalBlueberries = 4;
		wantBlueberries = 0;
	}

	public void UpdateNextStep(){
		if (totalBlueberries <= 1){
			wantBlueberries = 5;
		} else {
			wantBlueberries = 0;
		}
		totalBlueberries--;
	}
}
```

C (C.java)

```
public class C {

	public int totalCarrots = 4;
	public int wantCarrots = 0;

	public static void main(String [] args){
		// below is some sample code for if the user wanted to run this without the SRTI
		C thisClass = new C();
		for (int step = 0; step < 100; step++){
			if (step % 3 == 0){
				thisClass.UpdateNextStep();
			}
		}
	}

	public void Initialize(){
		totalCarrots = 4;
		wantCarrots = 0;
	}

	public void UpdateNextStep(){
		if (totalCarrots <= 1){
			wantCarrots = 5;
		} else {
			wantCarrots = 0;
		}
		totalCarrots--;
	}
}
```

G (G.java)

```
public class G {
	public int totalApples = 100;
	public int totalBlueberries = 50;
	public int totalCarrots = 60;
	public int buyRequestApples = 0;
	public int buyRequestBlueberries = 0;
	public int buyRequestCarrots = 0;
	public int soldApples = 0;
	public int soldBlueberries = 0;
	public int soldCarrots = 0;
	float totalMoney = 0.00f;
	float priceApples = 1.00f;
	float priceBlueberries = 2.50f;
	float priceCarrots = 2.00f;

	public static void main(String [] args){
		// below is some sample code for if the user wanted to run this without the SRTI
		G thisClass = new G();
		for (int step = 0; step < 100; step++){
			thisClass.UpdateStep();
		}
		System.out.println("done.");
	}

	public void Init(){
		totalApples = 100; totalBlueberries = 50; totalCarrots = 60;
		buyRequestApples = 0; buyRequestBlueberries = 0; buyRequestCarrots = 0;
		soldApples = 0; soldBlueberries = 0; soldCarrots = 0;
		totalMoney = 0f; priceApples = 1f; priceBlueberries = 2.5f; priceCarrots = 2f;
	}

	public void UpdateStep(){
		if (buyRequestApples < totalApples) {
			totalApples = totalApples - buyRequestApples;
			soldApples = buyRequestApples;
			buyRequestApples = 0;
		} else {
			soldApples = totalApples;
			totalApples = 0;
			buyRequestApples = 0;
		}
		if (buyRequestBlueberries < totalBlueberries) {
			totalBlueberries = totalBlueberries - buyRequestBlueberries;
			soldBlueberries = buyRequestBlueberries;
			buyRequestBlueberries = 0;
		} else {
			soldBlueberries = totalBlueberries;
			totalBlueberries = 0;
			buyRequestBlueberries = 0;
		}
		if (buyRequestCarrots < totalCarrots) {
			totalCarrots = totalCarrots - buyRequestCarrots;
			soldCarrots = buyRequestCarrots;
			buyRequestCarrots = 0;
		} else {
			soldCarrots = totalCarrots;
			totalCarrots = 0;
			buyRequestCarrots = 0;
		}
		totalMoney = totalMoney + (priceApples * soldApples)
			+ (priceBlueberries * soldBlueberries) + (priceCarrots * soldCarrots);
	}
}
```

Below are the simulators written in the Matlab scripting language:

A (A.m)

```
classdef A < handle
     properties
          totalApples = 4
          wantApples = 0
     end

     methods
          function obj = A()

          end

          function Initialize(obj)
               obj.totalApples = 4;
               obj.wantApples = 0;
          end

          function UpdateNextStep(obj)
               if (obj.totalApples <= 1)
                    obj.wantApples = 5;
               else
                    obj.wantApples = 0;
               end
               obj.totalApples = obj.totalApples - 1;
          end
     end
end
```

B (B.m)

```
classdef B < handle
     properties
          totalBlueberries = 4
          wantBlueberries = 0
     end

     methods
          function obj = B()

          end

          function Initialize(obj)
               obj.totalBlueberries = 4;
               obj.wantBlueberries = 0;
          end

          function UpdateNextStep(obj)
               if (obj.totalBlueberries <= 1)
                    obj.wantBlueberries = 5;
               else
                    obj.wantBlueberries = 0;
               end
               obj.totalBlueberries = obj.totalBlueberries - 1;
          end
     end
end
```

C (C.m)

```
classdef C < handle
	properties
		totalCarrots = 4
		wantCarrots = 0
	end

	methods
		function obj = C()

		end

		function Initialize(obj)
			obj.totalCarrots = 4;
			obj.wantCarrots = 0;
		end

		function UpdateNextStep(obj)
			if (obj.totalCarrots <= 1)
				obj.wantCarrots = 5;
			else
				obj.wantCarrots = 0;
			end
			obj.totalCarrots = obj.totalCarrots - 1;
		end
	end
end
```

G (G.m)

```
classdef G < handle
	properties
		totalApples = 100
		totalBlueberries = 50
		totalCarrots = 60
		soldApples = 0
		soldBlueberries = 0
		buyRequestApples = 0
		buyRequestBlueberries = 0
		buyRequestCarrots = 0
		totalMoney = 0.00
		priceApples = 1.00
		priceBlueberries = 2.50
		priceCarrots = 2.00
	end

	methods
		function obj = G()

		end

		function Init(obj)
			obj.totalApples = 100;
			obj.totalBlueberries = 50;
			obj.totalCarrots = 60;
			obj.soldApples = 0;
			obj.soldBlueberries = 0;
			obj.soldCarrots = 0;
			obj.buyRequestApples = 0;
			obj.buyRequestBlueberries = 0;
			obj.buyRequestCarrots = 0;
			obj.totalMoney = 0f;
			obj.priceApples = 1f;
			obj.priceBlueberries = 2.5f;
			obj.priceCarrots = 2f;
		end

		function UpdateStep(obj)
			if (buyRequestApples < totalApples){
				totalApples = totalApples - buyRequestApples;
				soldApples = buyRequestApples;
				buyRequestApples = 0;
			} else {
				soldApples = totalApples;
				totalApples = 0;
				buyRequestApples = 0;
			}
			if (buyRequestBlueberries < totalBlueberries){
				totalBlueberries = totalBlueberries - buyRequestBlueberries;
				soldBlueberries = buyRequestBlueberries;
				buyRequestBlueberries = 0;
			} else {
				soldBlueberries = totalBlueberries;
				totalBlueberries = 0;
				buyRequestBlueberries = 0;
			}
			if (buyRequestCarrots < totalCarrots){
				totalCarrots = totalCarrots - buyRequestCarrots;
				soldCarrots = buyRequestCarrots;
				buyRequestCarrots = 0;
			} else {
				soldCarrots = totalCarrots;
				totalCarrots = 0;
				buyRequestCarrots = 0;
			}
			totalMoney = totalMoney + (priceApples * soldApples)
				+ (priceBlueberries * soldBlueberries) + (priceCarrots * soldCarrots);
		end
	end
end
```

Below are the simulators written in the NetLogo scripting language:

A (A.nlogo)

```
globals [totalApples wantApples]

to Initialize
      clear-all
      set totalApples 4
      set wantApples 0
end

to UpdateNextStep
      ifelse totalApples <= 1 [set wantApples 5] [set wantApples 0]
      set totalApples totalApples - 1
end
```

B (B.nlogo)

```
globals [totalBlueberries wantBlueberries]

to Initialize
      clear-all
      set totalBlueberries 4
      set wantBlueberries 0
end

to UpdateNextStep
      ifelse totalBlueberries <= 1 [set wantBlueberries 5] [set wantBlueberries 0]
      set totalBlueberries totalBlueberries - 1
end
```

C (C.nlogo)

```
globals [totalCarrots wantCarrots]

to Initialize
      clear-all
      set totalCarrots 4
      set wantCarrots 0
end

to Initialize
      clear-all
      set totalCarrots 4
      set wantCarrots 0
end

to UpdateNextStep
      ifelse totalCarrots <= 1 [set wantCarrots 5] [set wantCarrots 0]
      set totalCarrots totalCarrots - 1
end
```

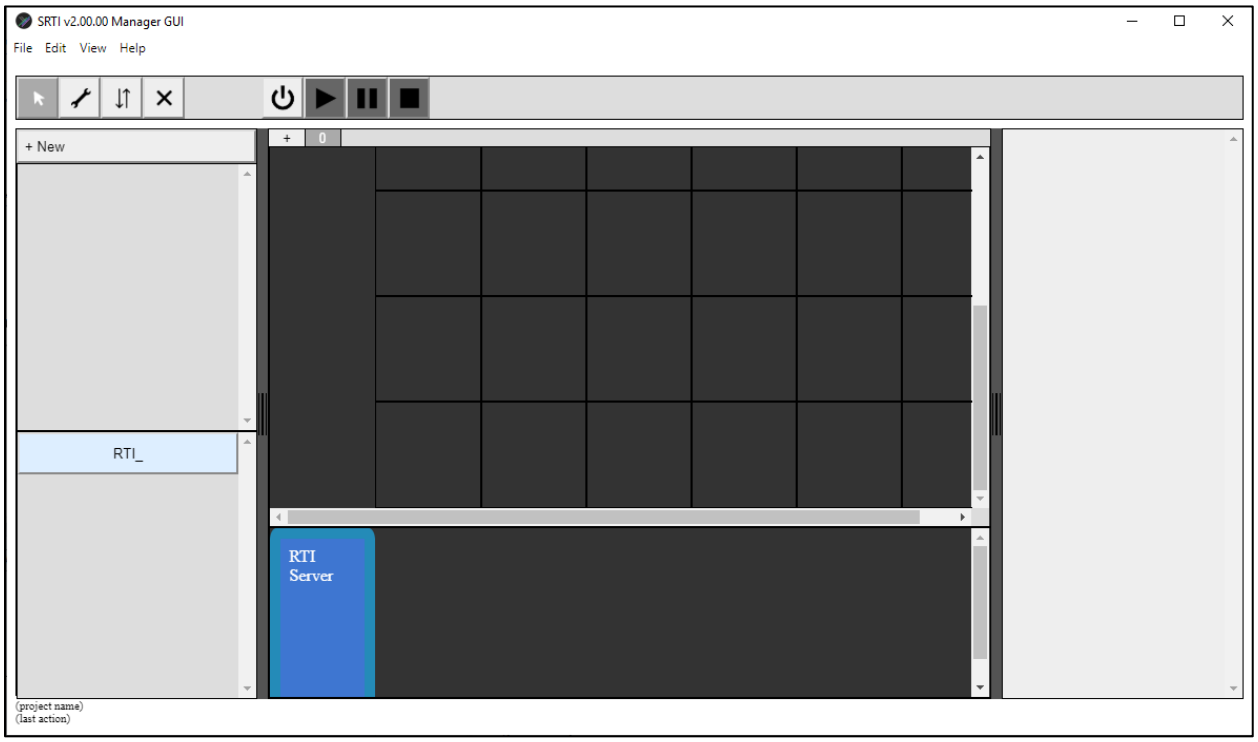
G (G.nlogo)

```
globals [totalApples totalBlueberries totalCarrots
      soldApples soldBlueberries soldCarrots
      buyRequestApples buyRequestBlueberries buyRequestCarrots
      totalMoney priceApples priceBlueberries priceCarrots]

to Init
      clear-all
      set totalApples 100
      set totalBlueberries 50
      set totalCarrots 60
      set soldApples 0
      set soldBlueberries 0
      set soldCarrots 0
      set buyRequestApples 0
      set buyRequestBlueberries 0
      set buyRequestCarrots 0
      set totalMoney 0
      set priceApples 1
      set priceBlueberries 2.5
      set priceCarrots 2
end

to UpdateStep
      if buyRequestApples < totalApples
      [
            set totalApples totalApples - buyRequestApples
            set soldApples buyRequestApples
            set buyRequestApples 0
      ]
      if buyRequestApples >= totalApples
      [
            set soldApples totalApples
            set totalApples 0
            set buyRequestApples 0
      ]
      if buyRequestBlueberries < totalBlueberries
      [
            set totalBlueberries totalBlueberries - buyRequestBlueberries
            set soldBlueberries buyRequestBlueberries
            set buyRequestBlueberries 0
      ]
      if buyRequestBlueberries >= totalBlueberries
      [
            set soldBlueberries totalBlueberries
            set totalBlueberries 0
            set buyRequestBlueberries 0
      ]
      if buyRequestCarrots < totalCarrots
      [
            set totalCarrots totalCarrots - buyRequestCarrots
            set soldCarrots buyRequestCarrots
            set buyRequestCarrots 0
      ]
      if buyRequestCarrots >= totalCarrots
      [
            set soldCarrots totalCarrots
            set totalCarrots 0
            set buyRequestCarrots 0
      ]
      set totalMoney totalMoney + (priceApples * soldApples) + (priceBlueberries * soldBlueberries) + (priceCarrots *
soldCarrots)
end
```

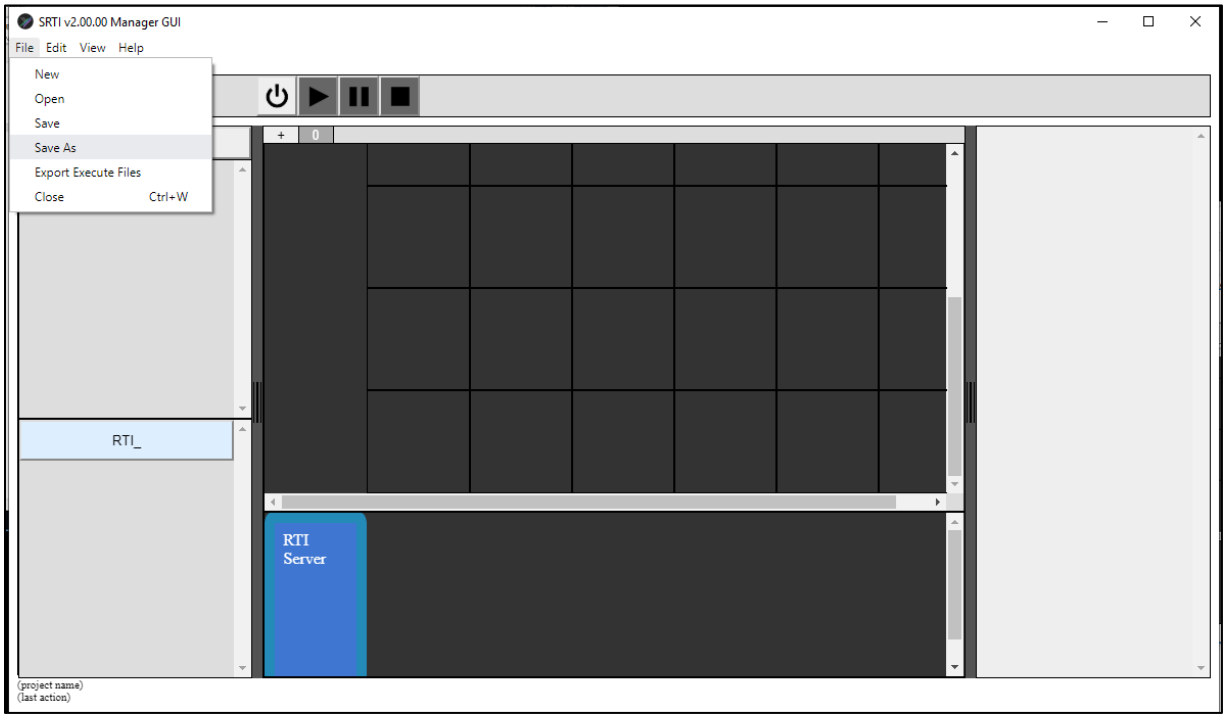
Regarding these languages: the Java simulators would have to be compiled into a .jar file for use. However, the Matlab and NetLogo simulators would not need to be pre-compiled. Also, the user's machine must have the Java Runtime Environment and Matlab to run corresponding simulators, but a copy of NetLogo is included inside the SRTI NetLogo Wrapper, so installing NetLogo is not required (and would not be used anyway by the SRTI Wrapper).

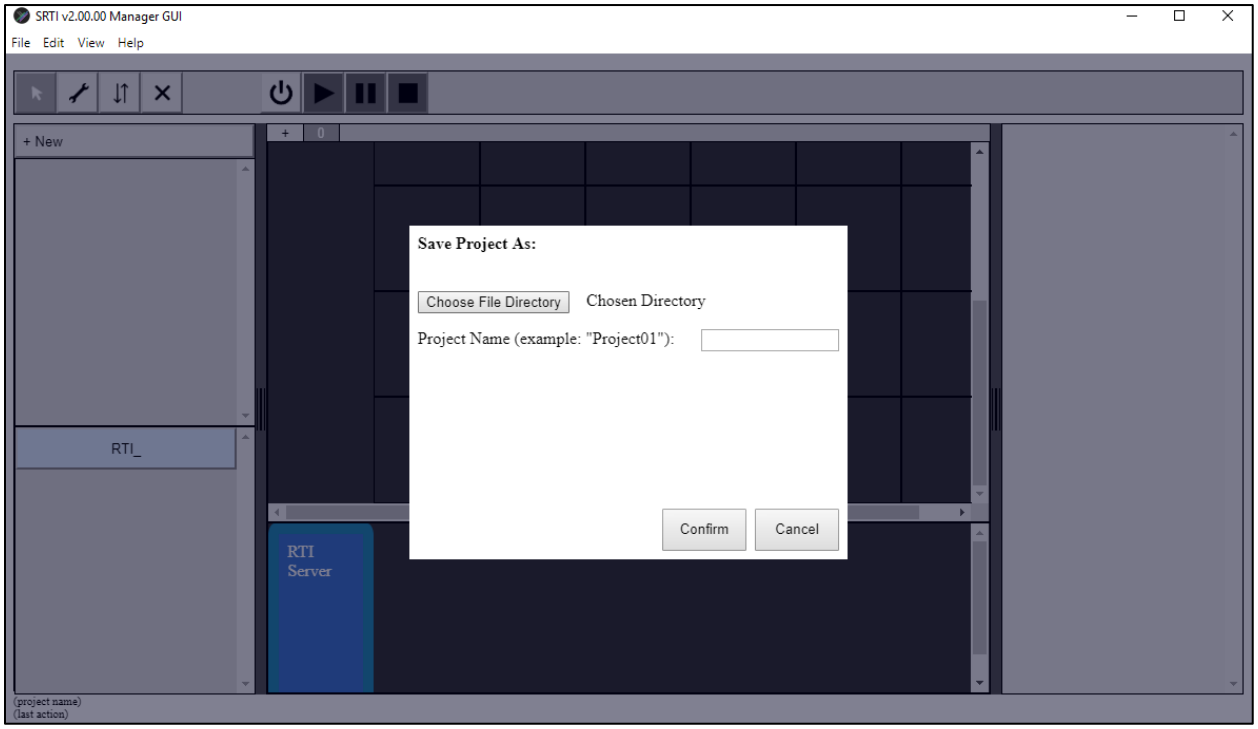
iii. Using SRTI v2 GUI

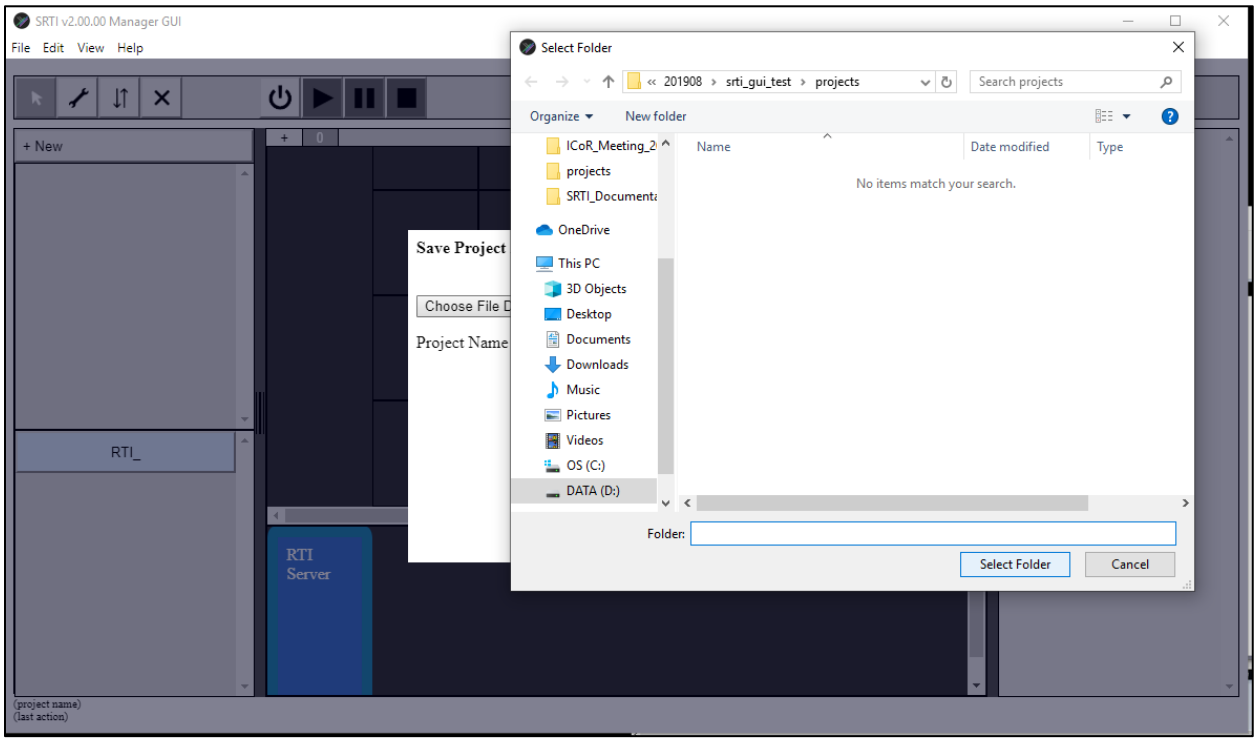
1) Open SRTI v2 GUI .exe.

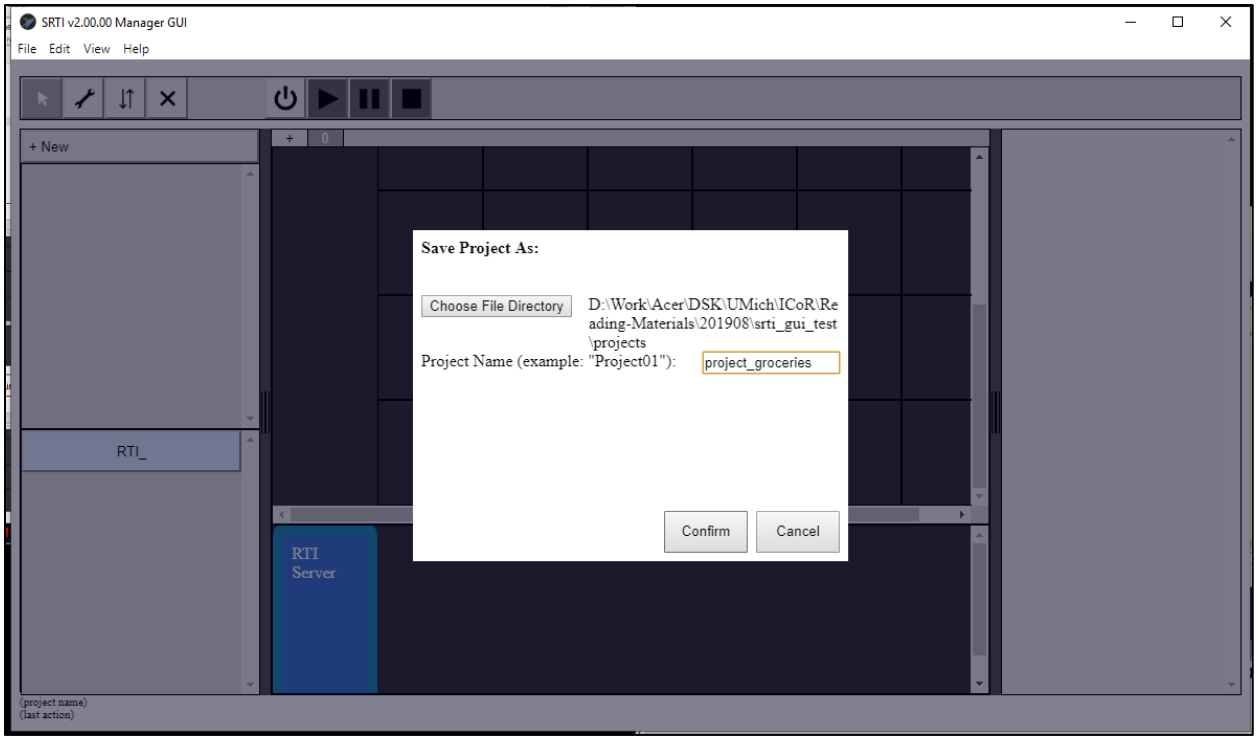


2) Save this project. We'll call it "project_groceries". Set the folder location, project name, and click "Confirm" when finished.

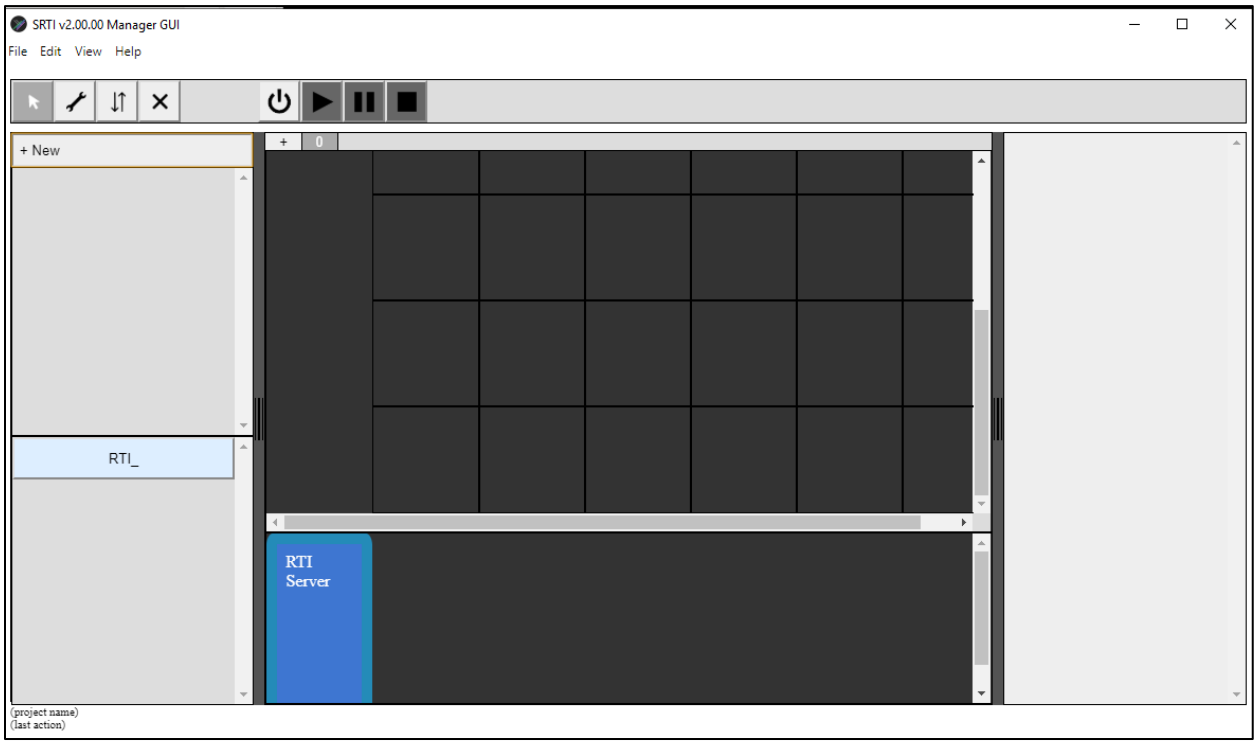



SRTI v2.00.00 Manager GUI
File Edit View Help
+ New
RTI_
RTI Server
Save Project As:
Choose File Directory
Chosen Directory
Project Name (example: "Project01"):
Confirm
Cancel
(project name)
(last action)

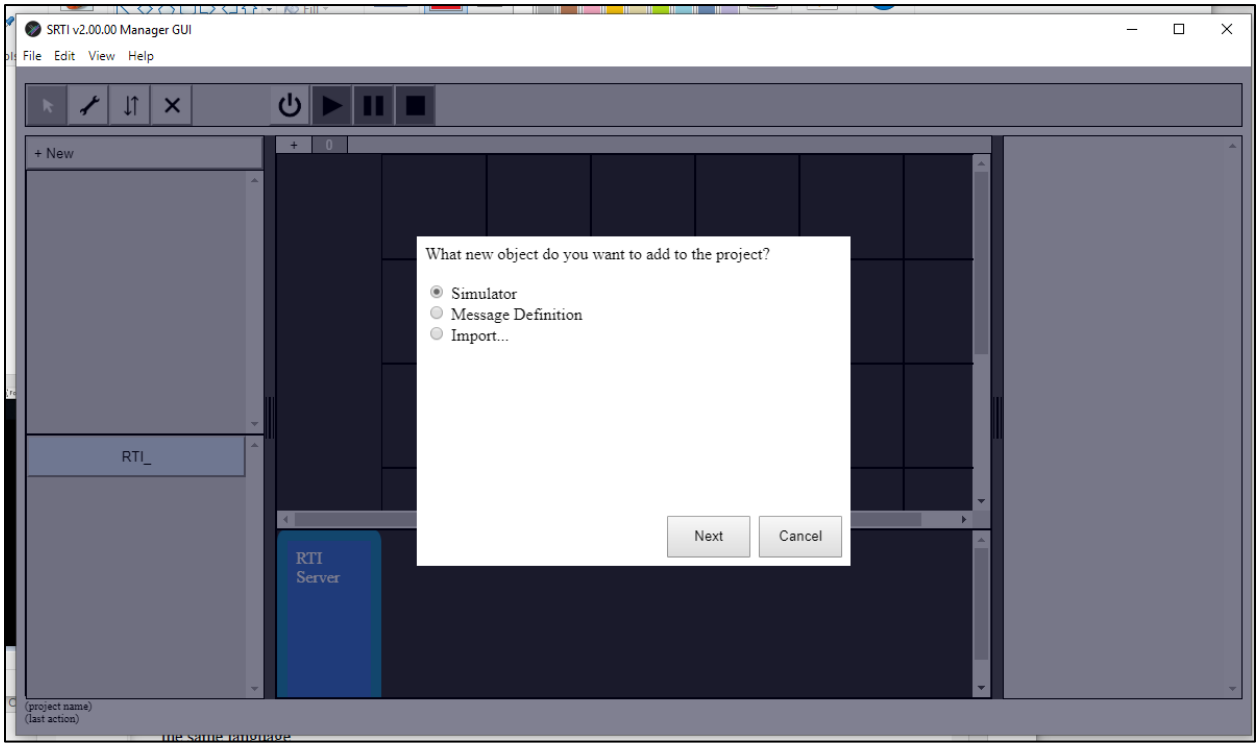

SRTI v2.00.00 Manager GUI
File Edit View Help
+ New
RTI_
RTI Server
Save Project
Choose File D
Project Name
Select Folder
201908 › srti_gui_test › projects
Search projects
Organize
New folder
ICoR_Meeting_2
projects
SRTI_Documenta
OneDrive
This PC
3D Objects
Desktop
Documents
Downloads
Music
Pictures
Videos
OS (C:)
DATA (D:)
Name
Date modified
Type
No items match your search.
Folder:
Select Folder
Cancel
(project name)
(last action)



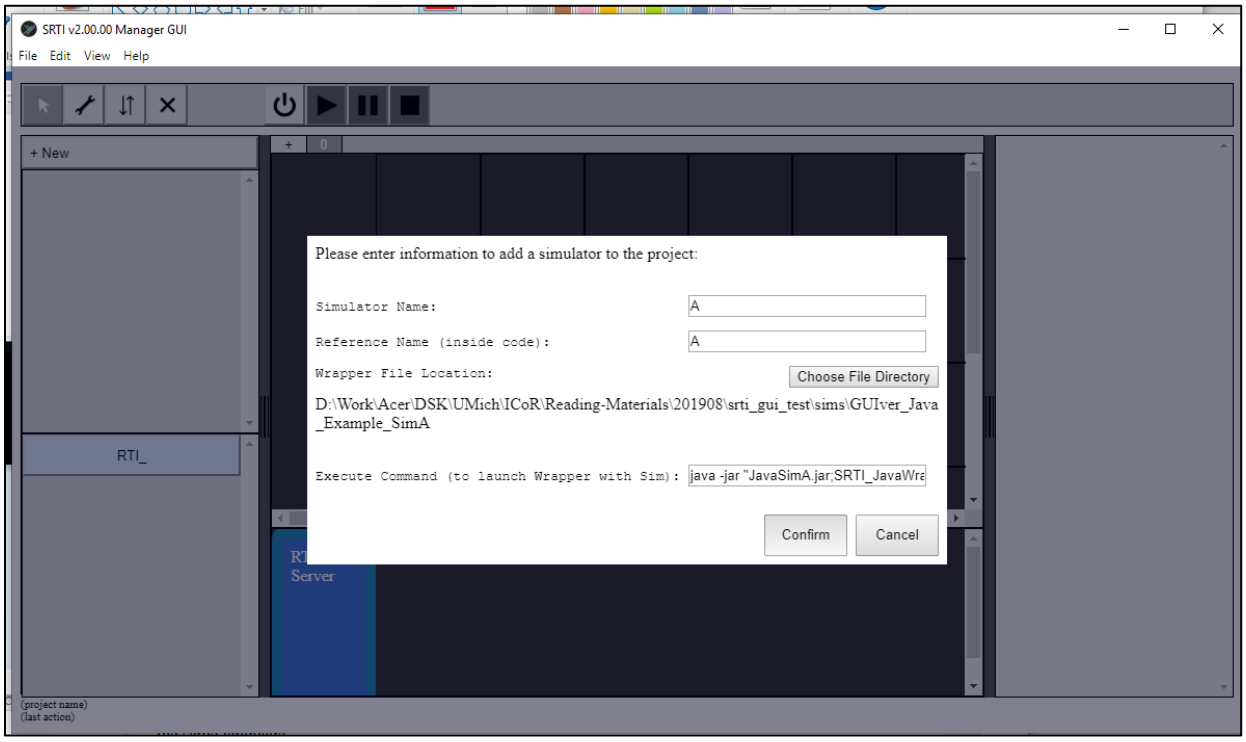
3) We need to add a definition / reference to every simulator in our system. We'll start by defining "A." Click on the "+ New" button at the top of the Object List panel.



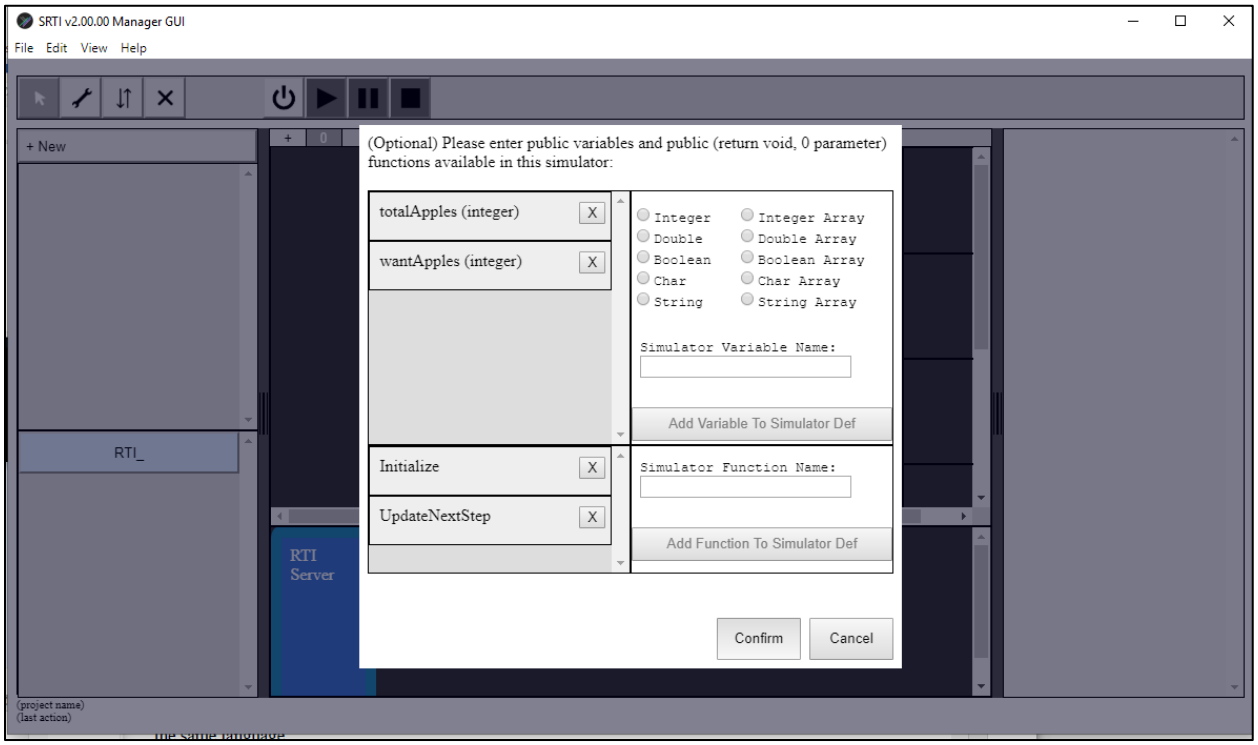
Select “Simulator” and click “Next.”



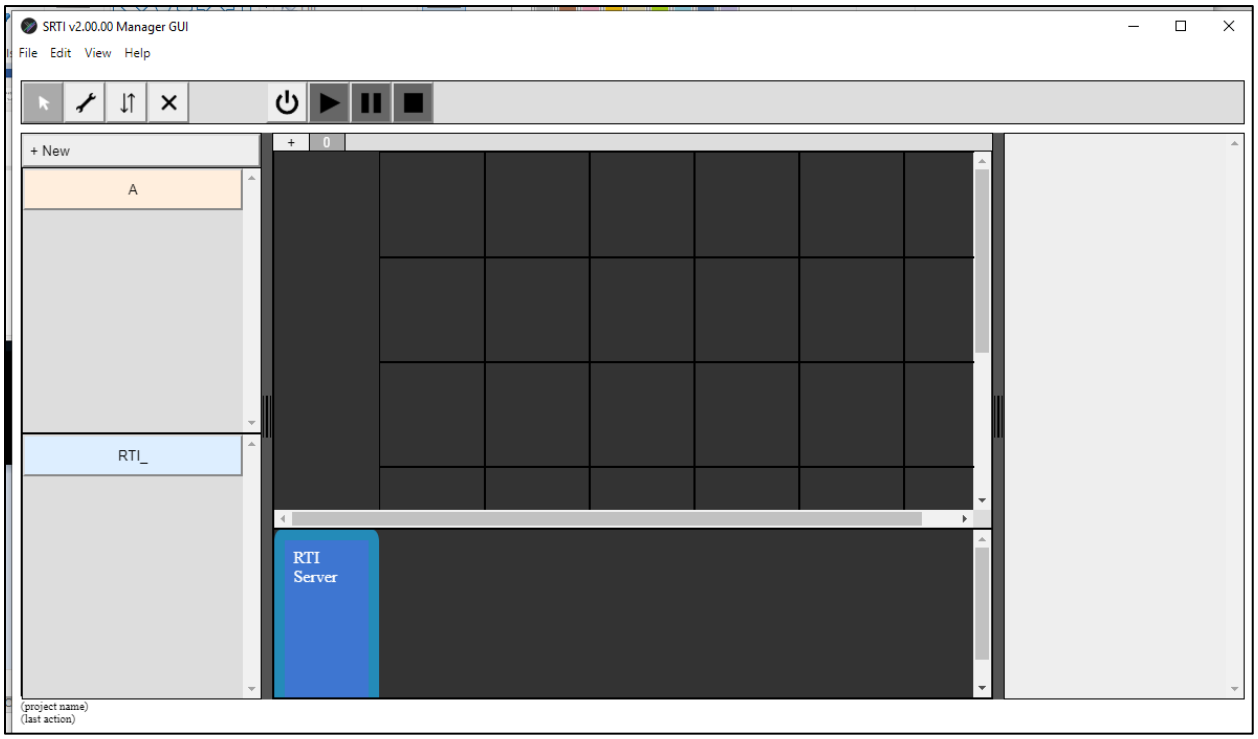
Fill in details, including “Name,” “Reference Name” (real name of class file), Wrapper/Simulator file location, and a command-line-style command to execute the Wrapper with the Simulator. More details can be found in separate documentation for SRTI v2.00.00.



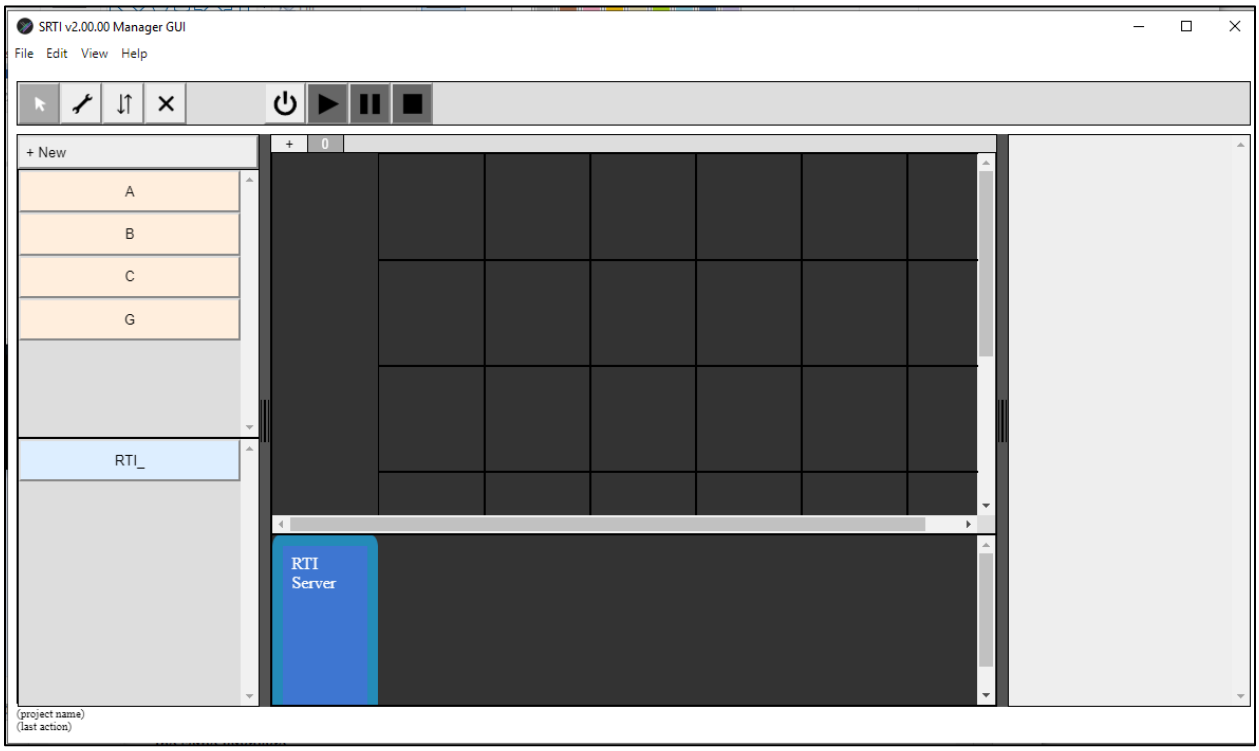
In the dialog box asking for public variables and functions inside Simulator A, enter the names and types of each variable and function to look like the following. Click “Confirm” to finish.



When finished, you should see Simulator “A” added to the Simulator Object List panel.

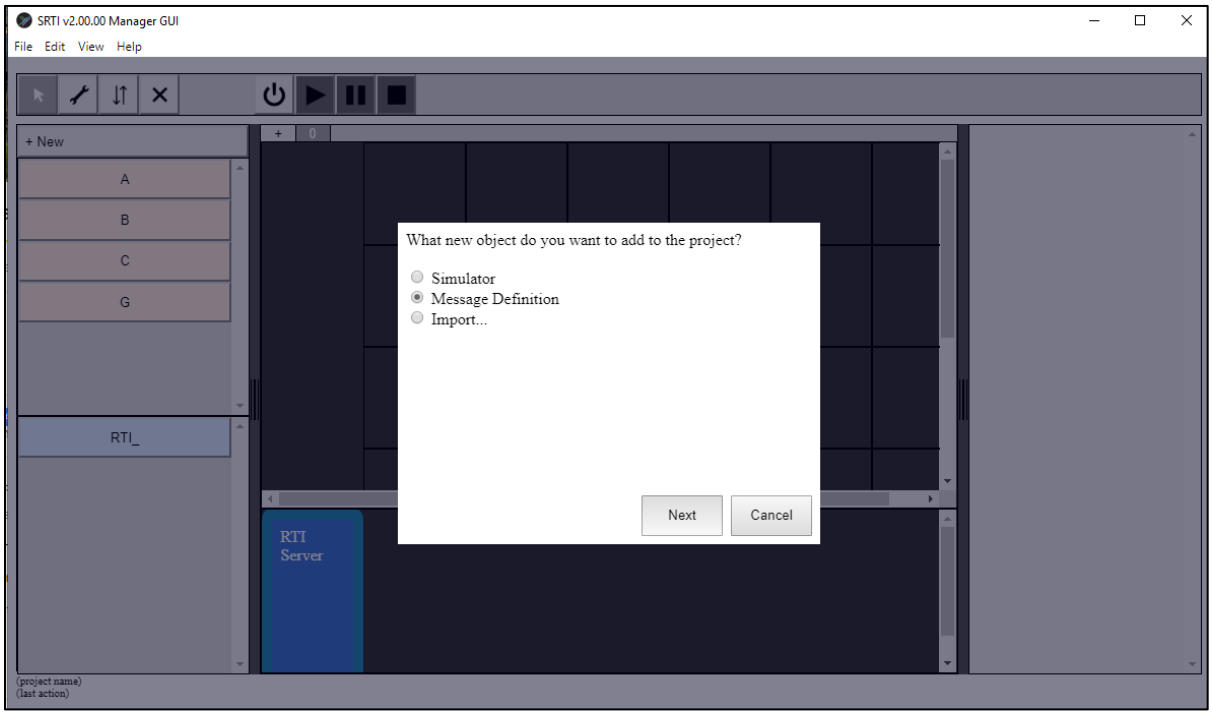


4) Repeat Step 3) to add simulators “B,” “C” and “G.” When finished, there should be 4 simulators in the Simulator Object List Panel (now would be a good time to “Save” the project too!).

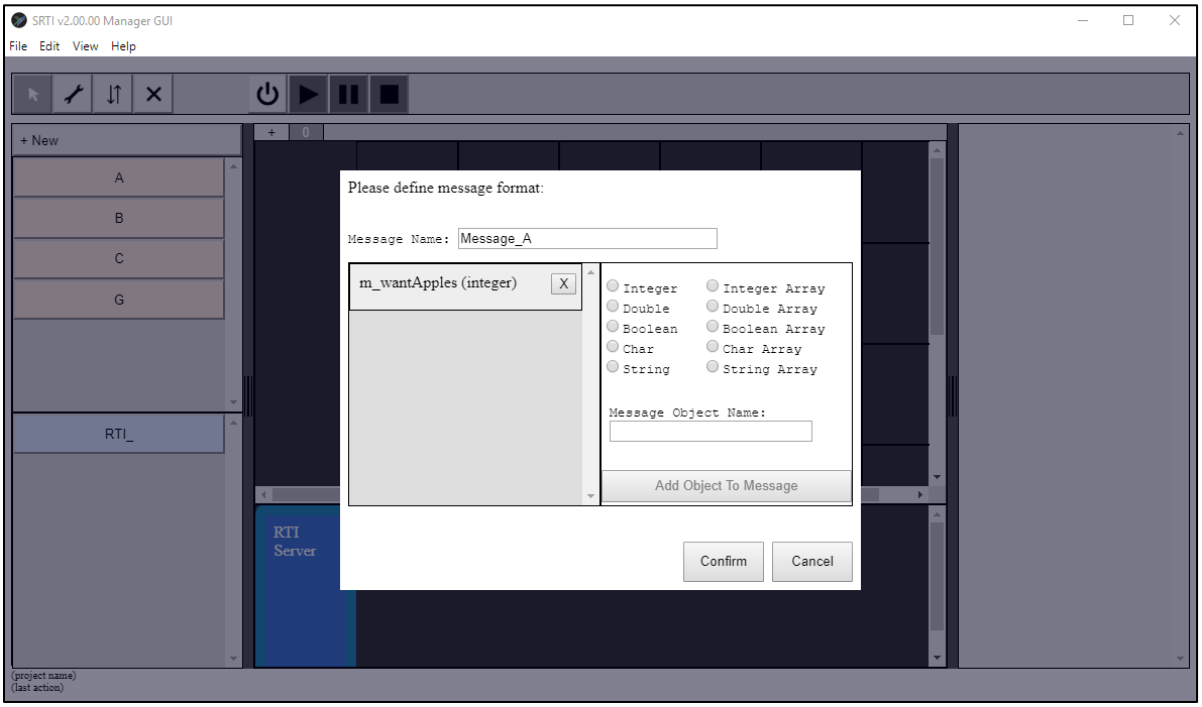


5) We need to add a definition for every message in our system.

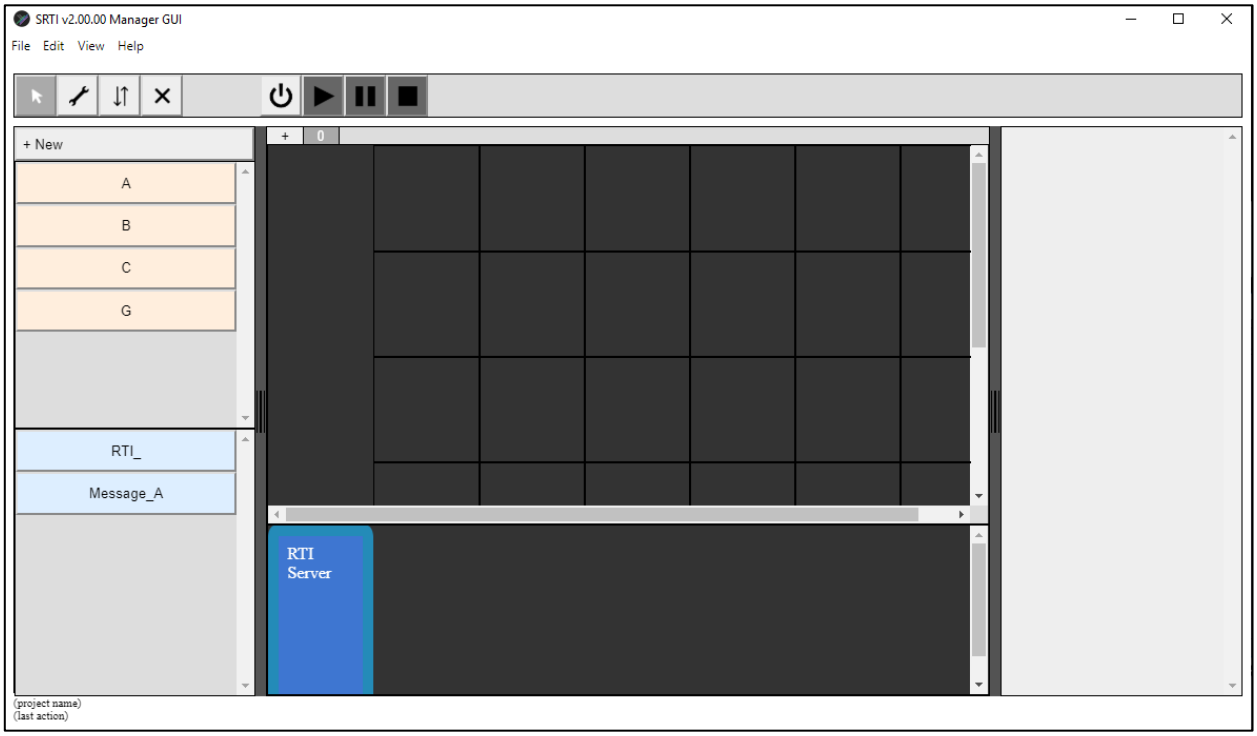
Click the "+ New" button at the top of the Object List Panel. Select "Message Definition" and click "Next."



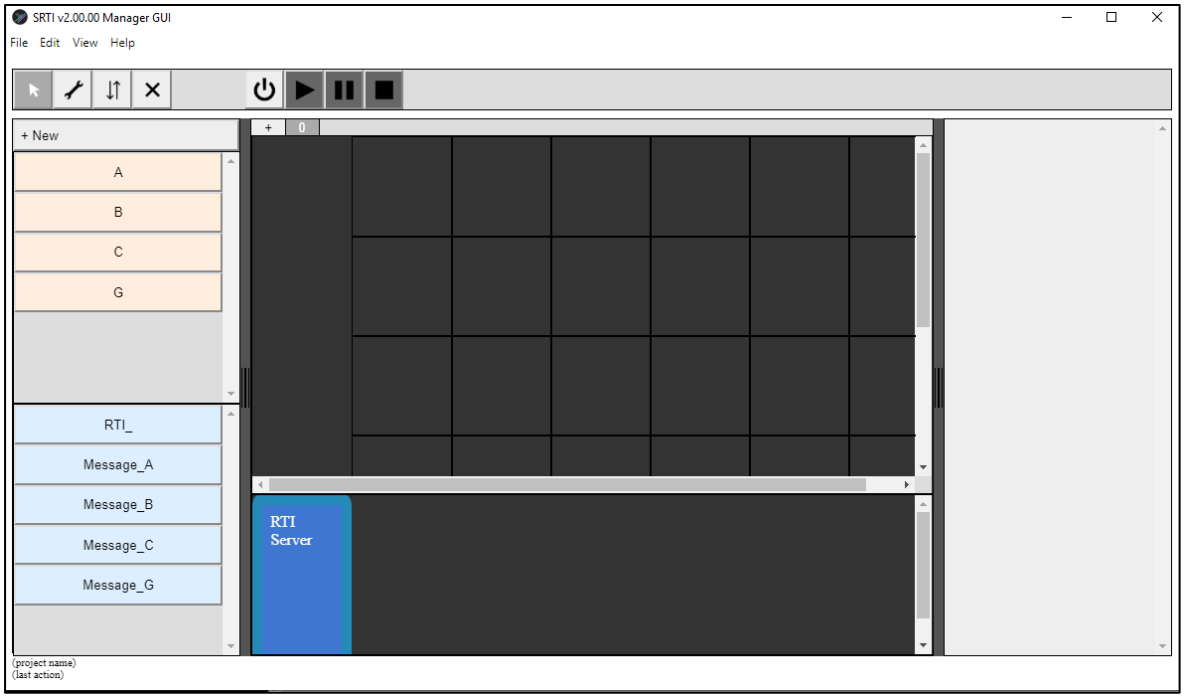
We'll assume that every simulator produces 1 message. Let's start with "Message_A." Give it this name and give it 1 Integer variable. Remember, we only care about the public variables that other simulators will want to know (and we can edit this later if we have to).



When finished, click the “Confirm” button.
You should see the Message added to the Message Object List Panel.

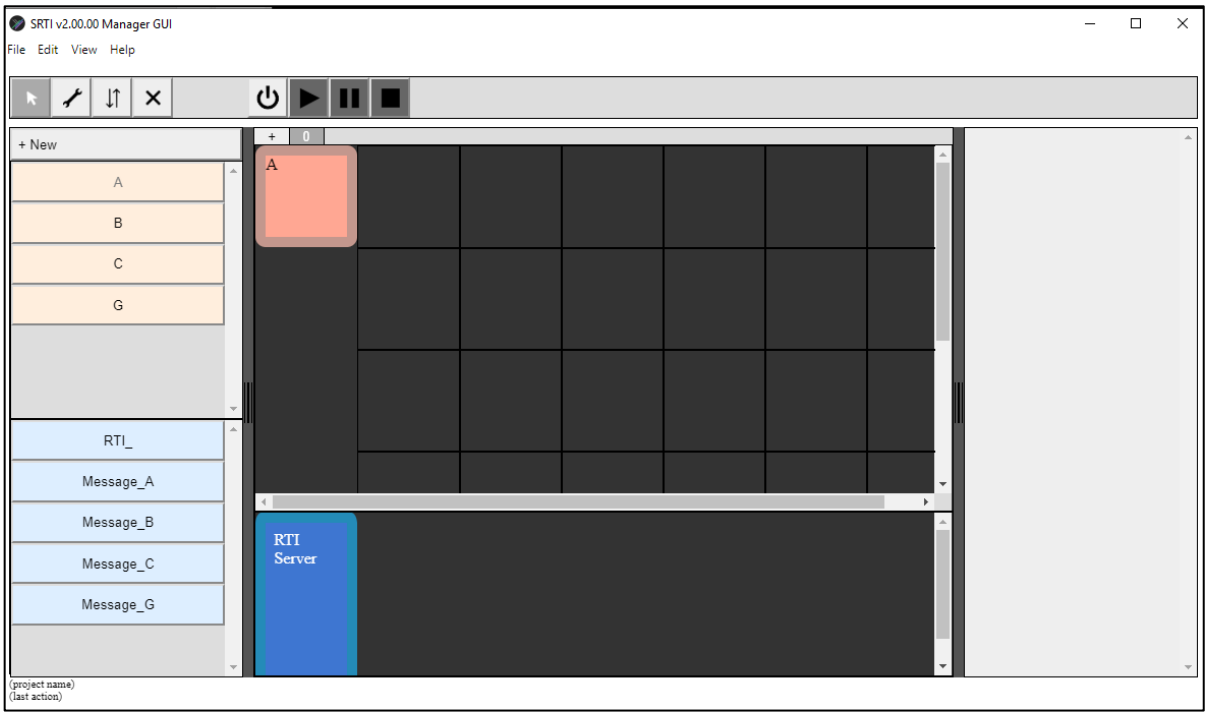


6) Repeat Step 5) to add 3 more messages to the project. In total, I added the following: “Message_A” (integer – m_wantApples), “Message_B” (integer – m_wantBlueberries), “Message_C” (integer – m_wantCarrots), “Message_G” (integer – m_soldApples, integer – m_soldBlueberries, integer – m_soldCarrots).

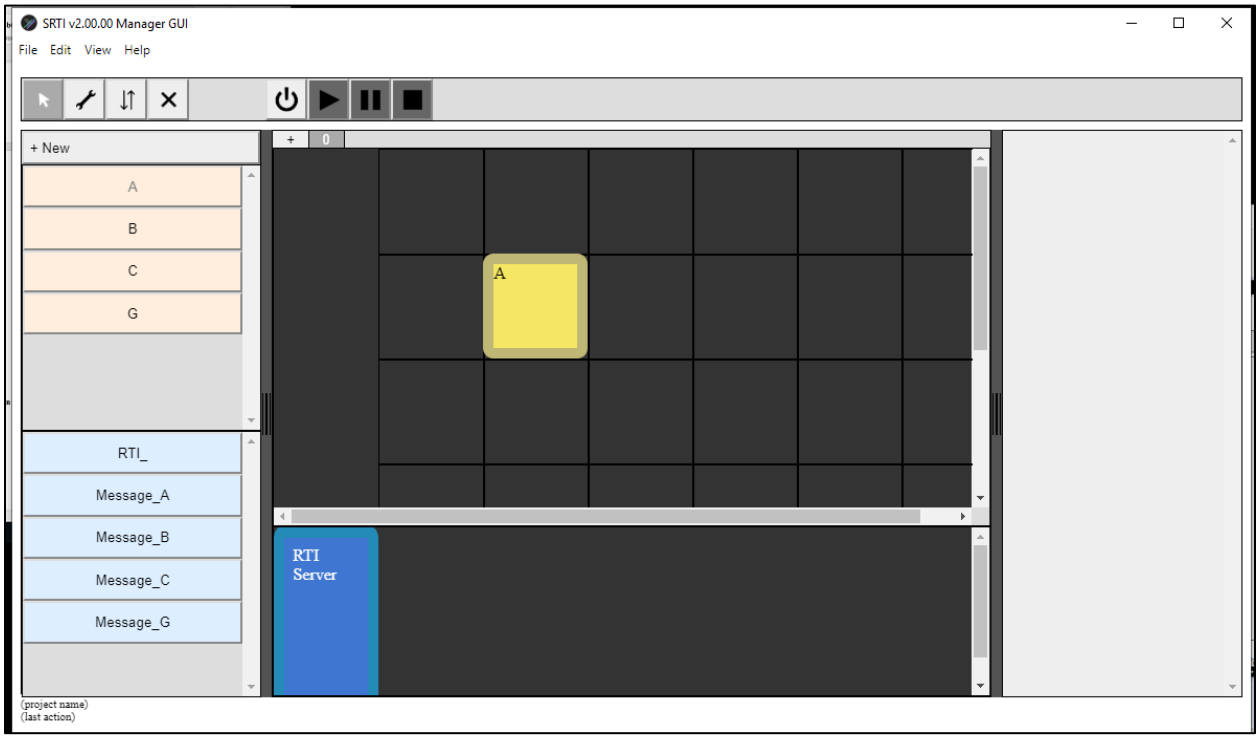


7) Now that the objects are prepared, we can add them to the canvas.

While in “Select” mode, click on the “A” simulator in the Simulator Object List Panel. It should appear in the top left corner of the upper canvas.

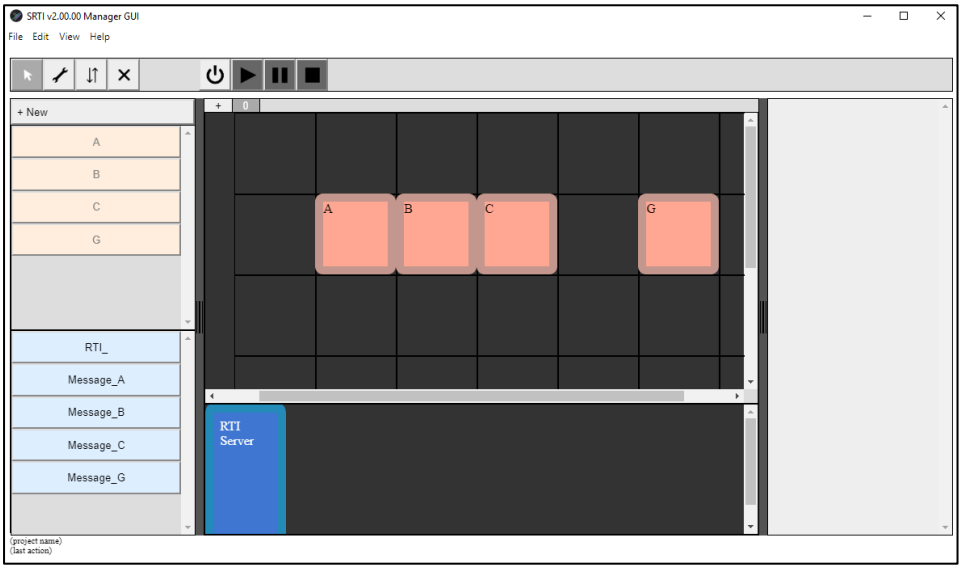


Click once on the Simulator object on the canvas again, and it will follow your mouse cursor around. Click a second time and the Simulator object will be released. You can do this to move the simulator on the canvas as you like.

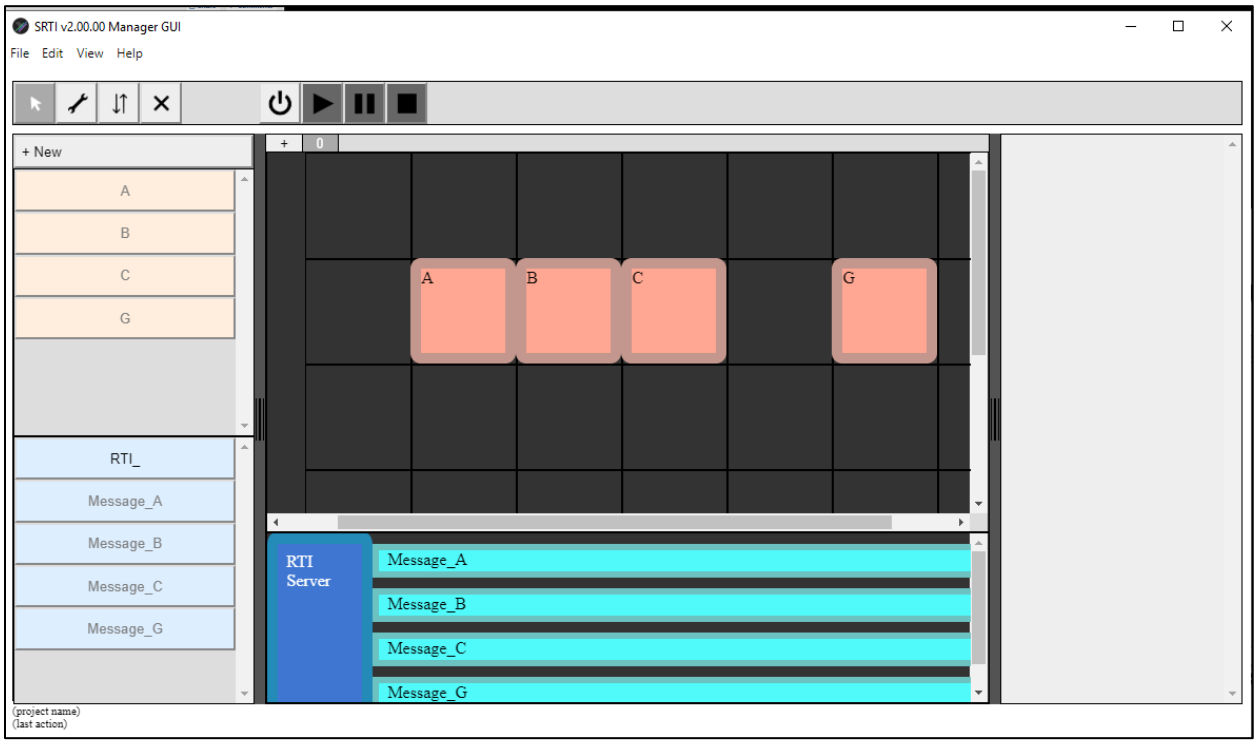


Generally, this is done for organizational purposes for the user, although the vertical placement relative to other simulators does have a meaning: simulators would be executed in "order" from the top to the bottom. For example, multiple simulators on the same vertical position would run at the same time, but a simulator above another would finish a step first before those below it.

Finish placing the other simulators, to have 4 simulator objects on the canvas.



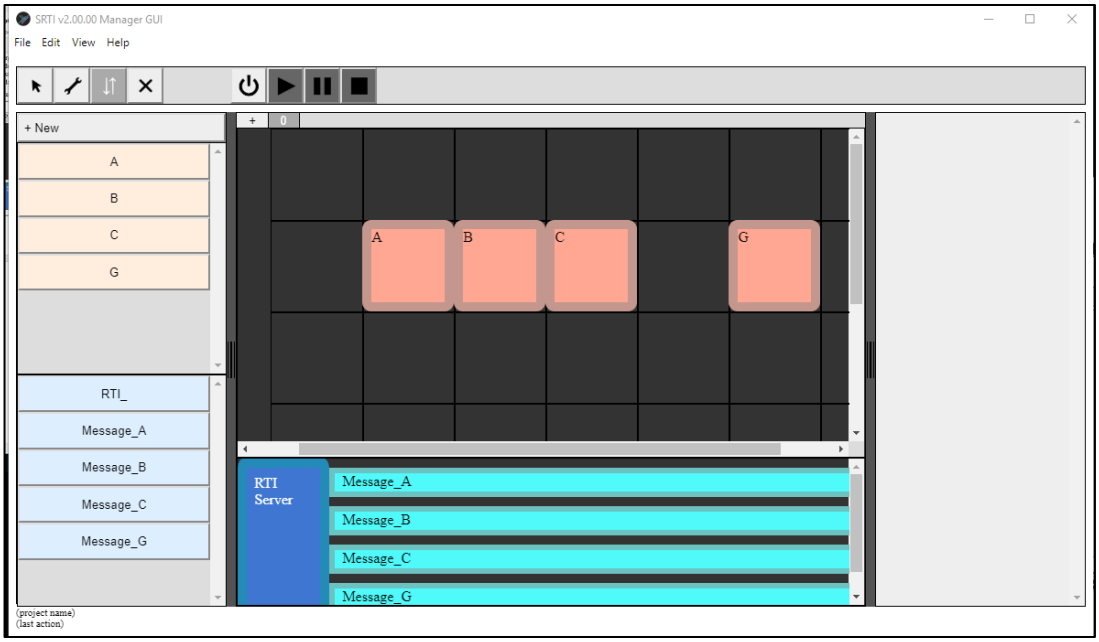
8) Place the message objects on the canvas. The process is similar to Step 7), but the user isn't able to reposition the messages in a certain order on the canvas; because the order is irrelevant to its execution.
   Now is also a good time to save the project.



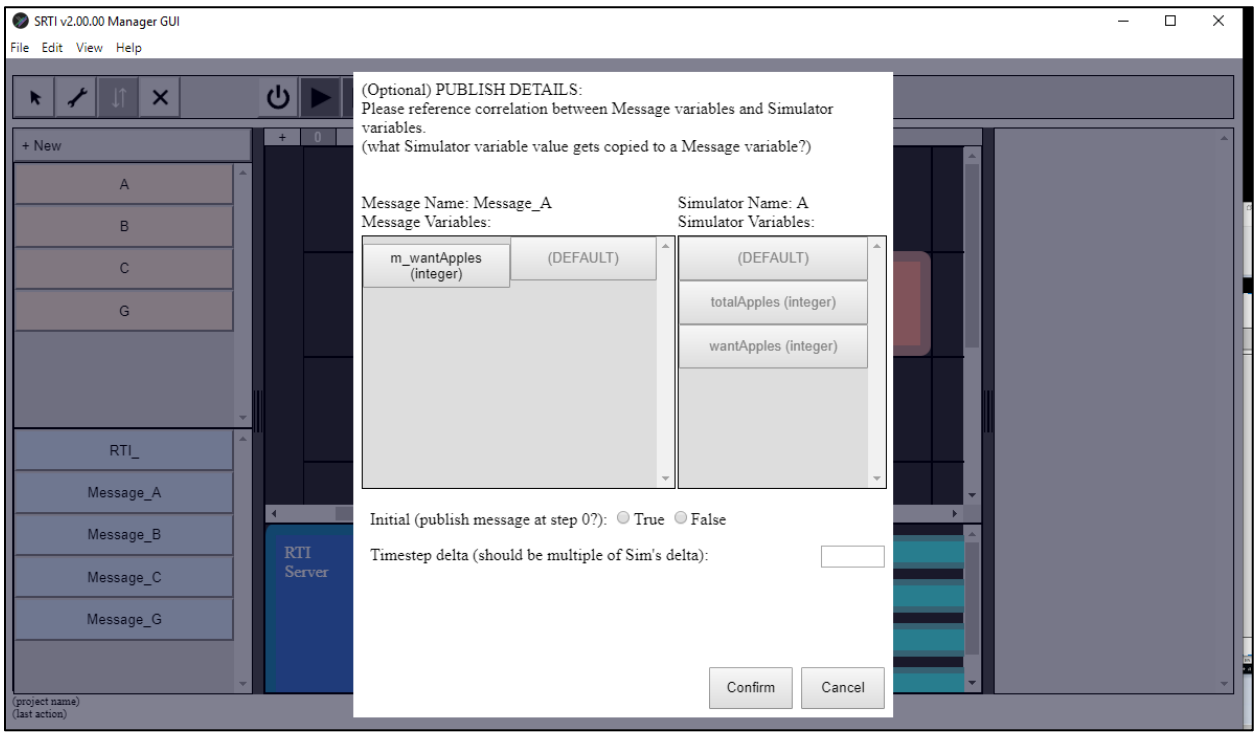
9) Next, we have to set "subscribe" and "publish" connections between each simulator and the messages.

Change the mode to "publish/subscribe" by clicking the 3rd button in the top-left "Action Toggle" sections.

While in this mode, first click on a simulator, then a message, to set a "publish" connection. Or, click on a message, then a simulator, to set a "subscribe" connection.

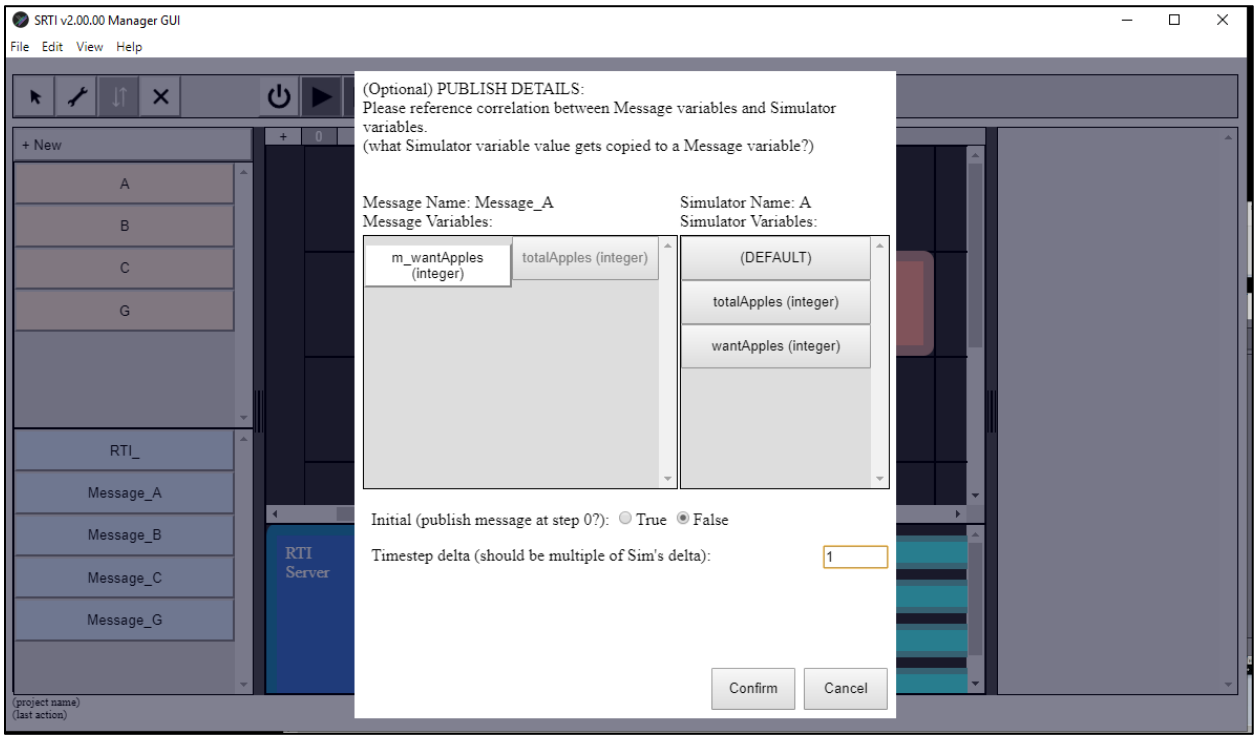


We'll start by describing how "A" publishes to "Message_A." Click on "A" on the canvas, then click on "Message_A". The connection should already be set, but a dialog box will appear to allow you to configure additional details about how variables correspond to each other.

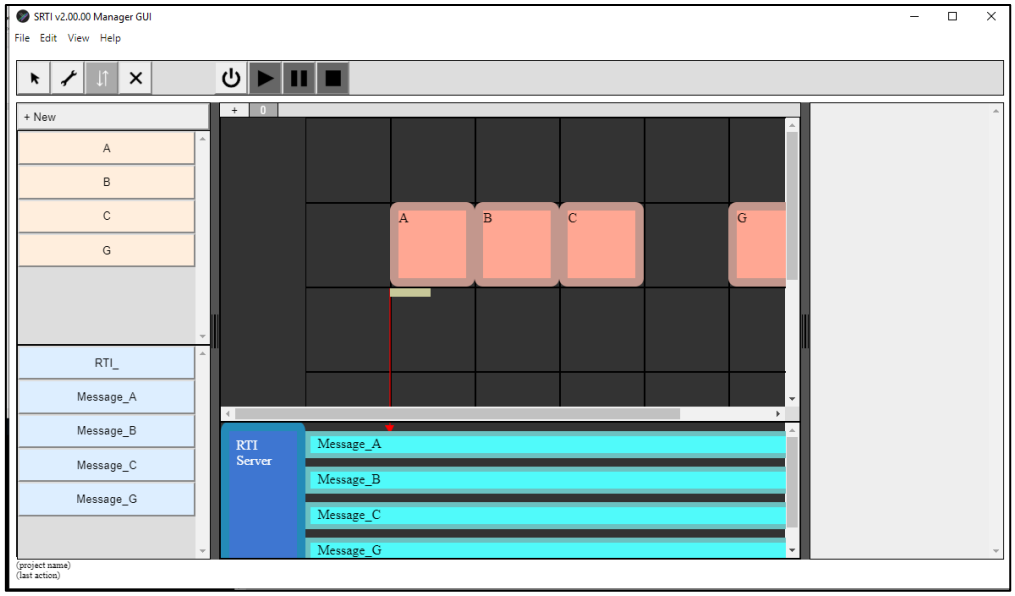


Click on “m_wantApples” and then “wantApples,” to say we want the value of the variable “wantApples” inside the simulator “A” to be copied into “m_wantApples” in the message “Message_A.”

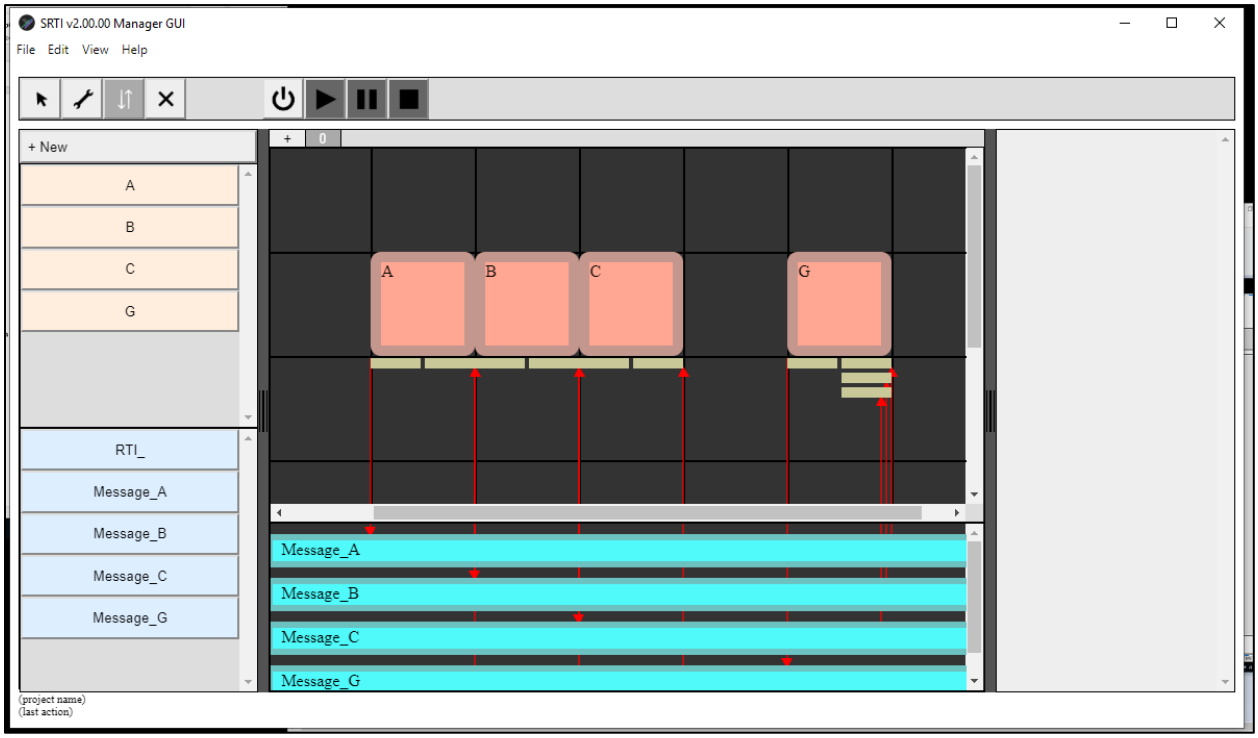
Also, set “Initial” (whether or not the message should only be published at the beginning) to “false,” and “Timestep delta” to 1. Click “Confirm” to finish.



You should see a small square underneath the simulator “A,” with an arrow that goes from it to “Message_A” in the below canvas. In the “Configure” mode, you can click on this small square to edit these publish details.



10) Repeat Step 11) to set publish and subscribe details for all the simulators.

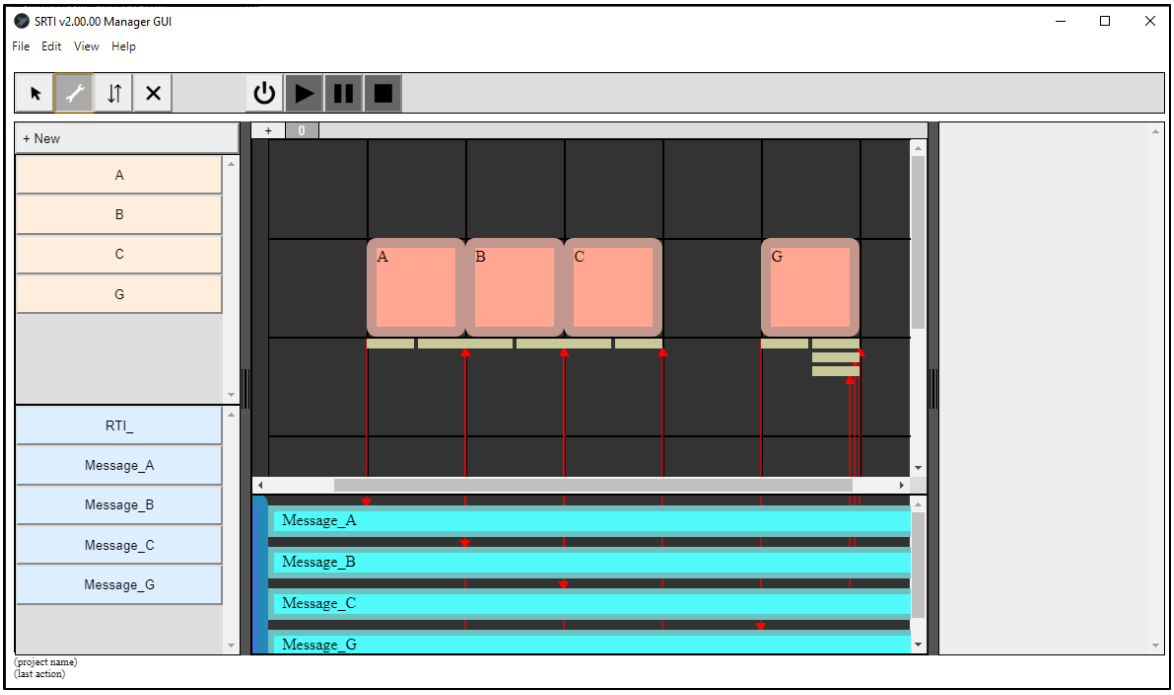


11) Save the project: the UI design is done!

… well, almost. Remember, we haven't yet described how the SRTI will know that the system should stop, or what functions should be called to simulate logic for each simulator at each timestep.

Let's start by describing "Initialize" and "Simulate" functions for each simulator.

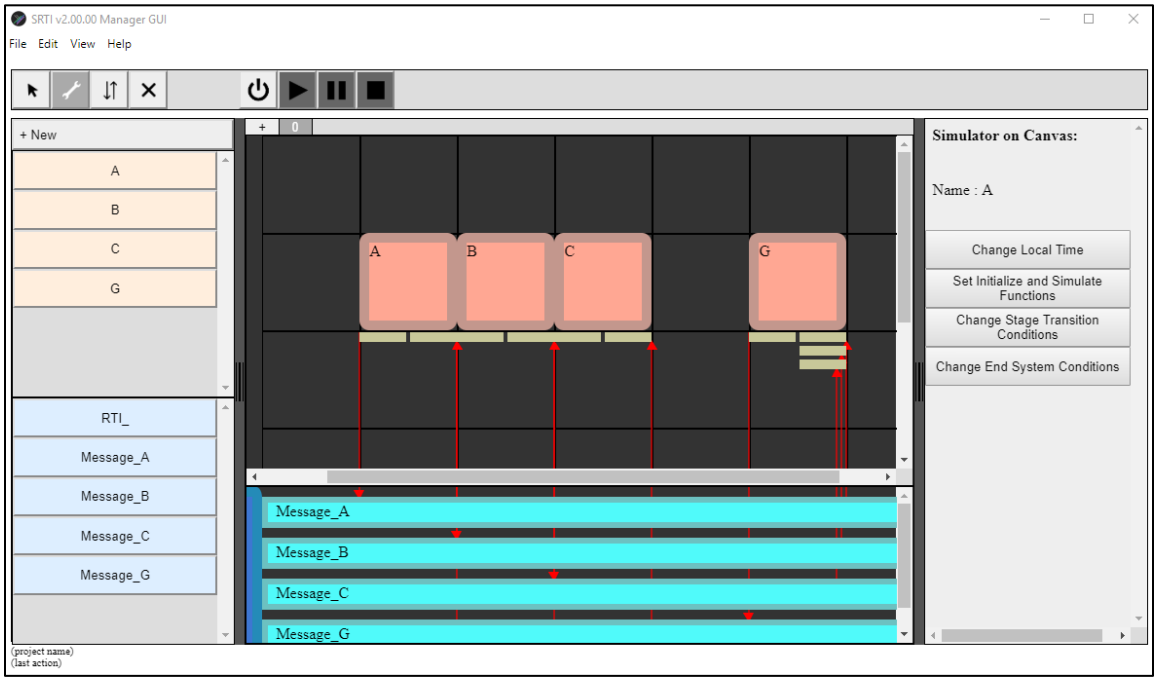
Click on the 'configure' button, the 2rd along the Action Toggle Buttons, to change the selection mode.



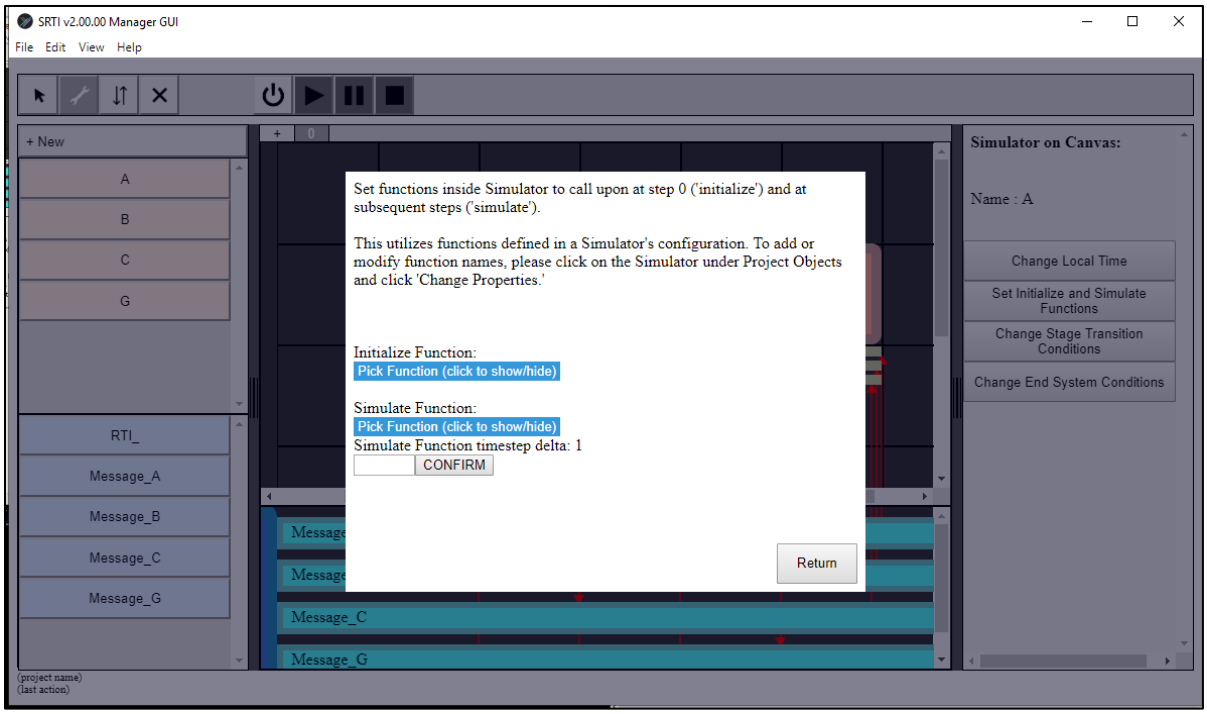
In this mode, try clicking on objects from both the Object List and the Canvas (Simulator, Message, Publish-Object, Subscribe-Object). You should see different text and buttons appear on the Inspector Panel on the right.

Changing properties of a Simulator or Message in the Object List will change global properties for instances of it on the canvas. Changing properties of objects on the canvas will change only that instance, which might be useful for custom examples in between different stages or projects.

Let's first set "Initialize" and "Simulate" functions for Simulator A. Click on "A" on the canvas.



In the Inspector Panel, click on “Set Initialize and Simulate Functions.” A new window should appear.



You can use the dropdown menus to choose a pre-defined function that exists inside “A” as the Initialize or Simulate functions (or you can choose to leave these empty). For “A,” we’ll set Initialize to “Initialize” and Simulate to “UpdateNextStep.”

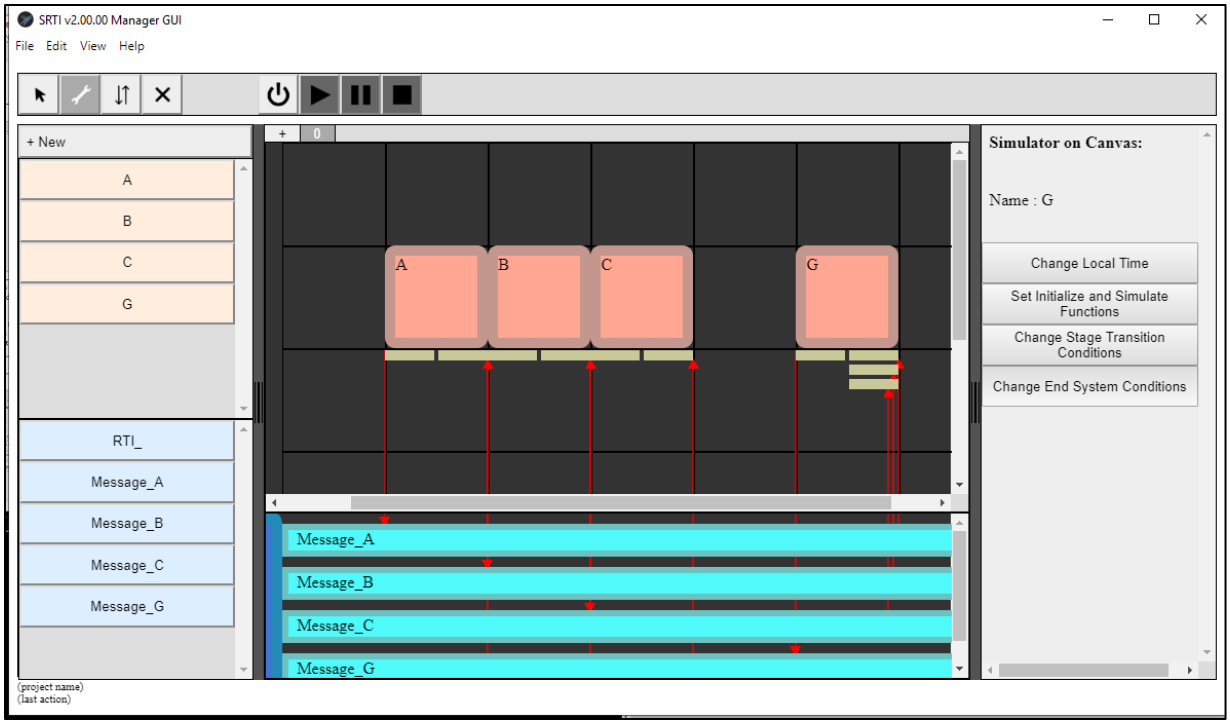
Click the Return button to close the window.

12) Repeat step 11) for simulators “B,” “C,” and “G.”

13) While not essential, we can also set the “EndConditions” we want for this system.

To keep things simple, we’ll use “G” as the only example to close the system: if “G” runs out of Apples, Blueberries or Carrots, the system will close.

In “Configure” mode, click on “G” on the canvas and click “Change End System Conditions.”

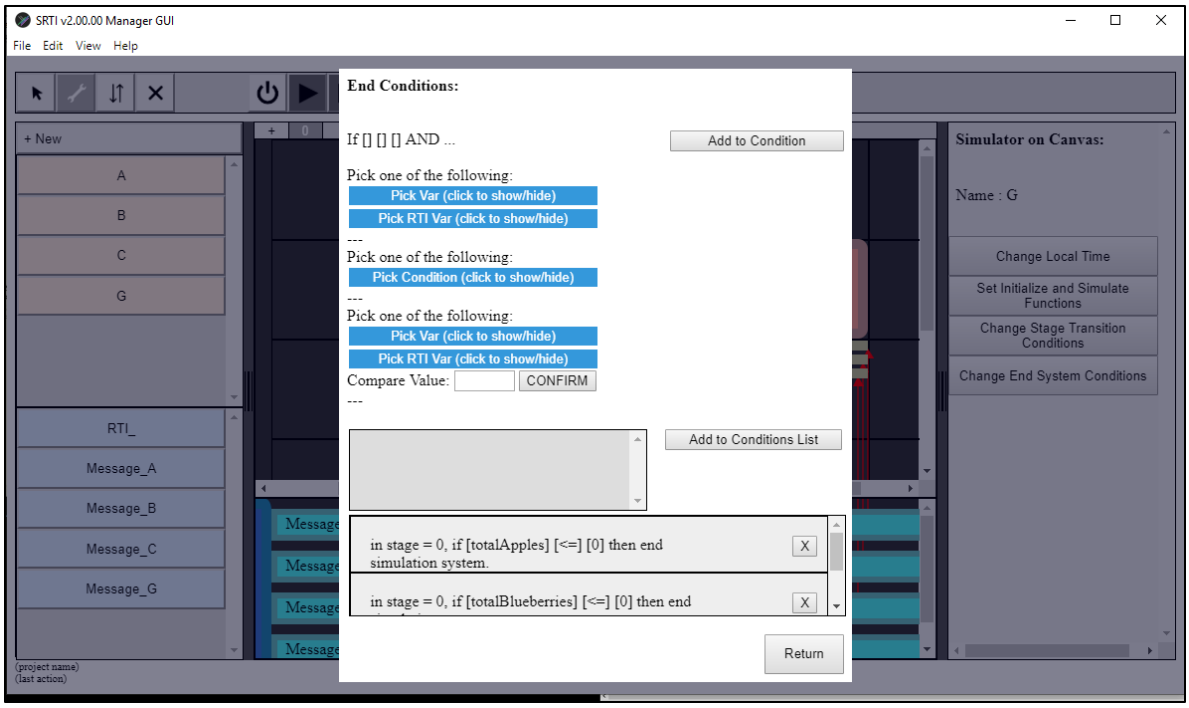


A window will popup with dropdown menus to choose a variable from inside the simulator, a comparison operator, and a new variable (or value) to compare it to.

This window has 2 different lists to add a condition to. What's the difference? When you create a condition and click on "Add to Condition," it goes to the first list. You can add other conditions to this same list: this is equivalent to saying ALL of these conditions must be true to end the system. When you are satisfied, you can click on "Add to Conditions List" to add the resulting condition to the final list: there, if any one of the conditions is satisfied, the system will end. This allows some basic AND and OR operators to be utilized.

We'll set 3 separate conditions: check if "totalApples <= 0", if "totalBlueberries <= 0", or if "totalCarrots <= 0".

When finished, click on the "Return" button.



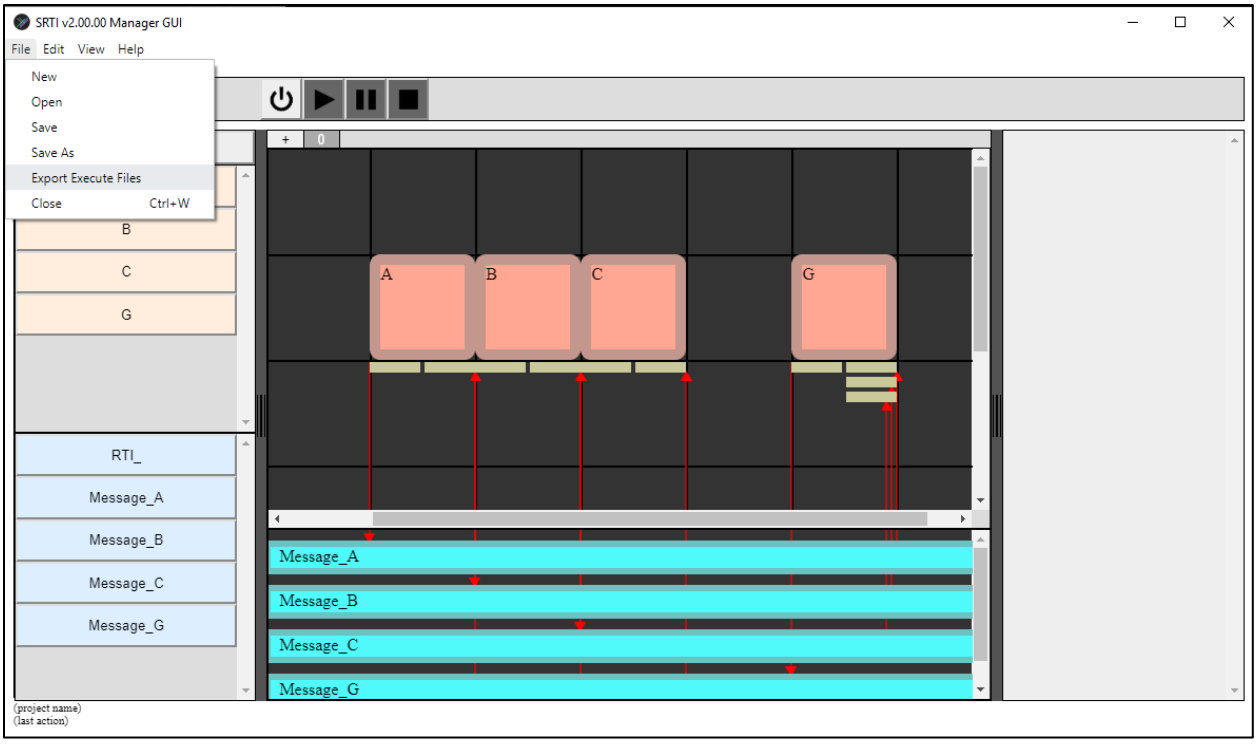
14) Before we finish, a brief word about a common user mistake:

In this example, we have the potential of a "deadlock." That's because "A," "B" and "C" are subscribed to a message from "G," and "G" is also subscribed to messages from "A," "B" and "C." If we were to run this without the correct settings, the system would never proceed past the first step: they would wait indefinitely for each other!

To resolve this, one of the messages must be sent first, even if the contents are basically default ("0"). We'll have "G" publish "Message_G" first. In "Configure" mode, click on the square that represents the publish details from "G" to "Message_G." In the Inspector Panel, click on "Change Publish Parameters." Double-check that "Initial" is set to "True," then click "Confirm" to close the window.

15) Save the project: NOW the UI design is done!

Remember that this is solely a visual description of the simulation execution system we want. To output the real Wrapper configuration files for use, click on "File -> Export Execute Files." This will save Wrapper files in the same folder location as each Wrapper/Simulator, as described when we first added a simulator to the project. Keep in mind that this may overwrite existing Wrapper configuration files.



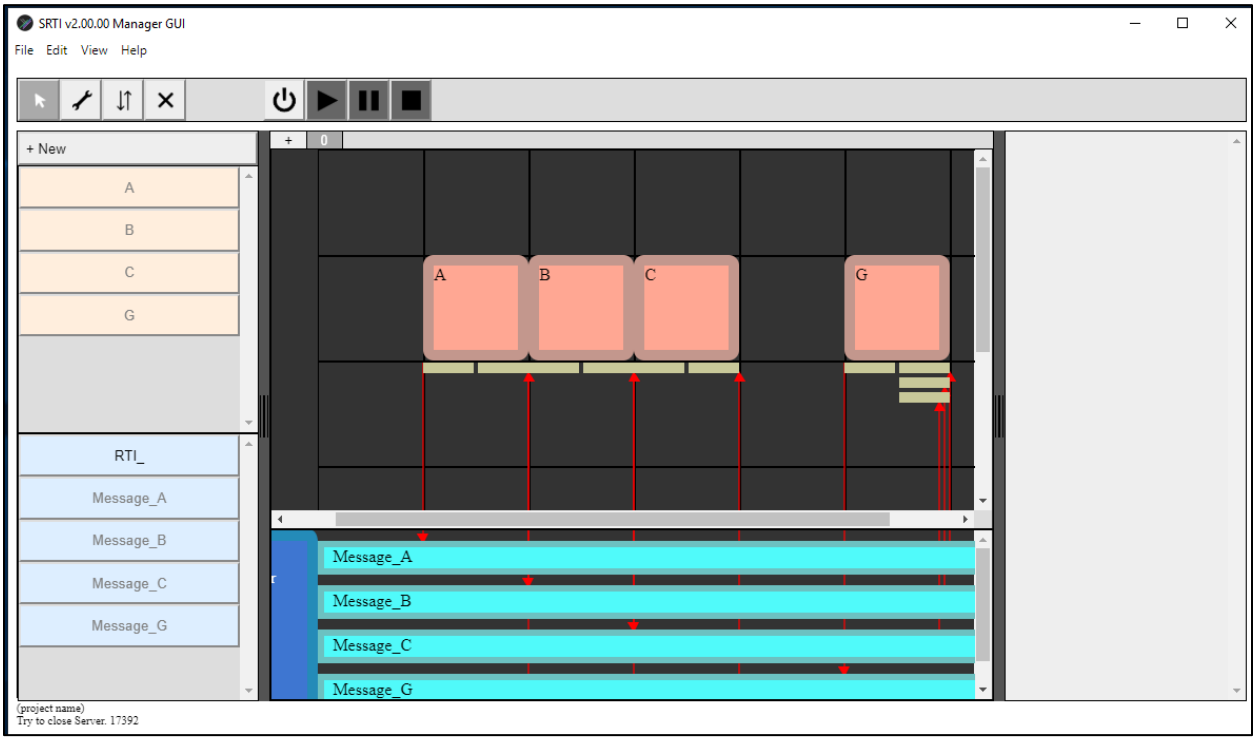
iv. Running the system using SRTI v2 GUI

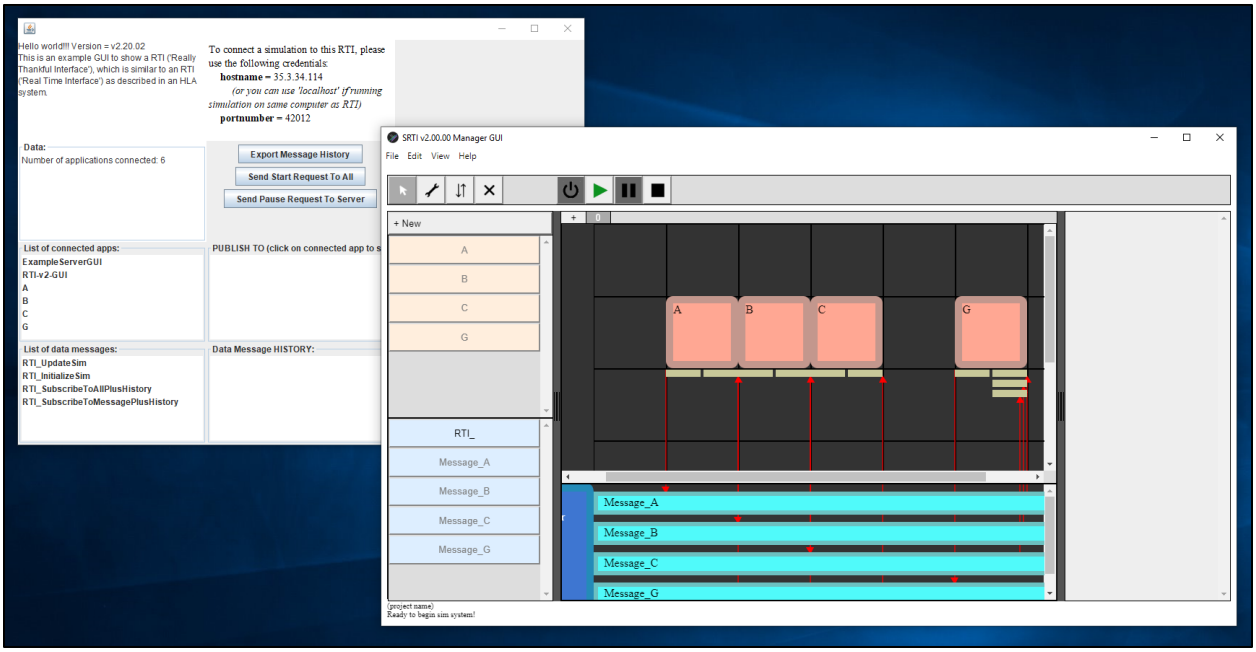
After clicking "Export Execute Files," the user can run the simulation system. This can either be done directly within the GUI, or separately using a command line prompt to open each simulator one by one. This section will explain how to do this within the GUI.

Click on the "Power On" button (in the Execute Buttons section). The GUI will open an instance of the RTI Server (with it's own GUI to signal that it is open), and a few seconds later, will open the individual simulators. There is an intentional wait-time between opening the RTI Server and the simulators (if the RTI Server isn't already open, an error will occur when their Wrappers try to connect to it). Allow at least 10 seconds for everything to finish opening.

At this point, if any error messages occur in popup windows, stop and try to figure out the cause (it may be an internal error with the GUI, or a configuration error with your project).

Notice that the Execute Buttons have changed: now the "Play" and "Stop" buttons are possible to click. As a reminder, "Stop" would close the RTI Server and the simulators, "Play" will start/resume the simulation system, and "Pause" will pause the simulation system.





Hit “Play” and see what happens. The RTI Server GUI should show messages updating, and after several seconds, should automatically disconnect the simulators. “Pause” and “Stop” should also function as expected.

After clicking “Stop,” if any error messages occur in popup windows, this is normal and expected behavior. Don’t worry about it.

Other future features for the SRTI v2 GUI include being able to display simulator variables and system timestep within it to better trace the system's execution outside the individual simulators.

Chapter 3 – Additional Notes

For assistance or questions, users can contact developer Andrew (Andy) Hlynka at ahlynka@umich.edu .