GUIDE

Advanced Scripting for Simulation

SPECINNOVATIONS.COM

INTRODUCTION

Innoslate's discrete event and Monte Carlo simulators require little scripting for most applications. The logic execution uses the decision points and Input/Output (I/O) triggers for basic problems, but for more complex simulations, you will want to learn how to add scripts, which will range from minimal to complex.

JAVASCRIPT BASICS[1]

JavaScript was initially created to "make web pages alive". The programs in this language are called scripts. They can be written right in a web page's HTML and run automatically as the page loads. Scripts are provided and executed as plain text. They don't need special preparation or compilation to run. The syntax for JavaScript is shown in Figure 1.

For adding to Innoslate scripts, note the use of "let" to define a variable. Innoslate script allows either "let" or "var" to define variables. Comments can also be added using two backslashes (//). Also, "print()" statements can easily be added. These statements print to the Innoslate simulator console.

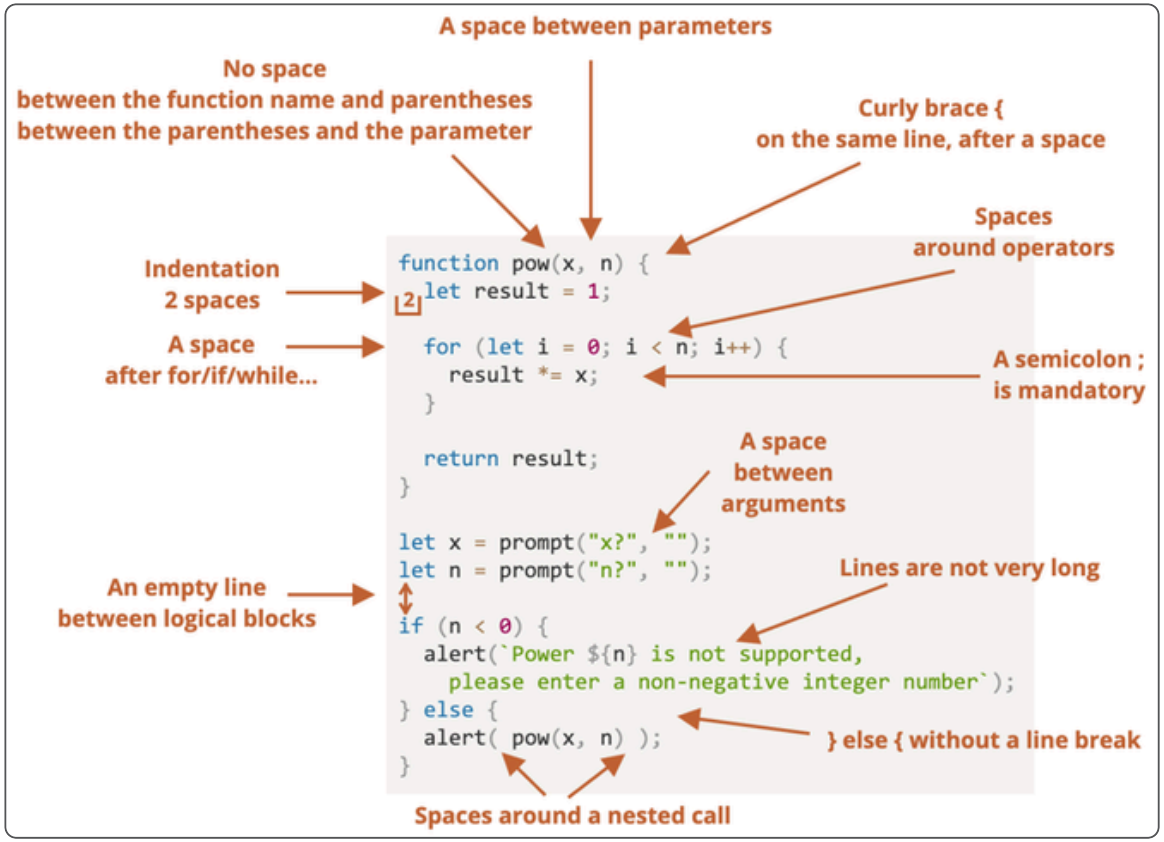


Figure 1. Script Syntax[2]

PRE-BUILT SCRIPTS

The OR and LOOP decision points (special cases of the Action) already come with pre-built scripts (Probability and Resource for both and Iterations for the Loop). These can be used without any scripting on your part. Just select the script type you want, adjust the values, and submit the script, then we generate the script automatically. The code behind each decision point's script type can be accessed by selecting 'Script' from the drop-down menu. An example of each is shown in the later sections below.

OR SCRIPT

The user selects "Or Probability" from the drop-down menu, as seen in Figure 2, then can adjust the branch probabilities, in this case making Yes 70% of the time and No 30%.

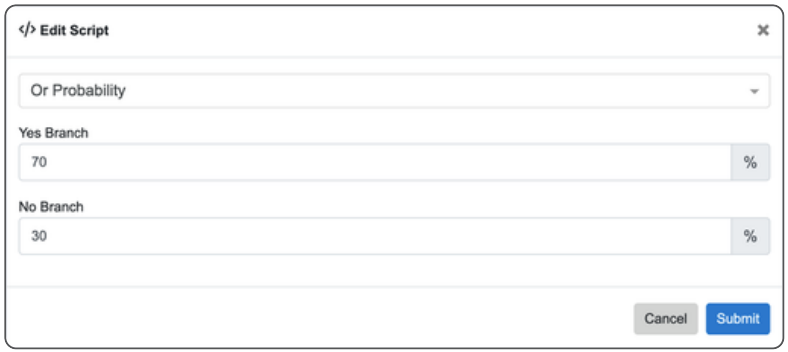


Figure 2. Or Probability User Interface

Figure 3. Script Generated for Or Probability

The script that is generated is shown in Figure 3. Note that the comment line at the top contains the information shown in the user interface above. The comment list each branches ID number with its corresponding assigned percentages. If changes are made it is recommend to either update assigned percentages or delete it.

OR RESOURCE SCRIPT

Figure 4 shows the user interface for the Or Resource script. A Resource entity must be created first and available in the database. The branch options include logical constructs and values.

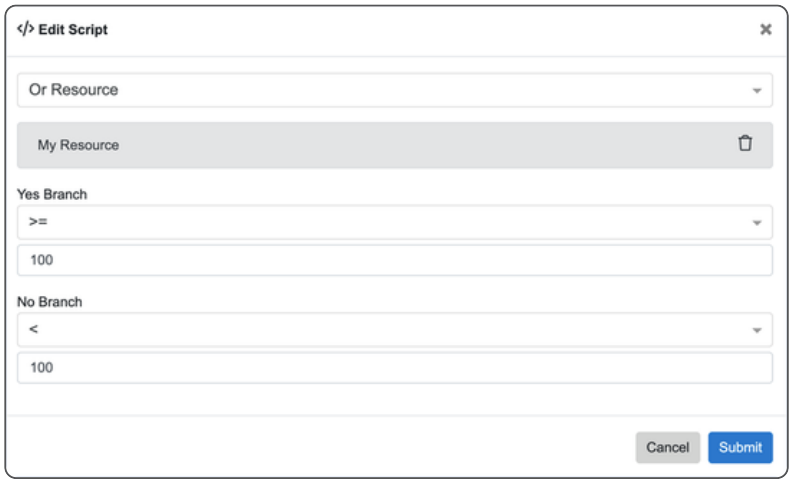
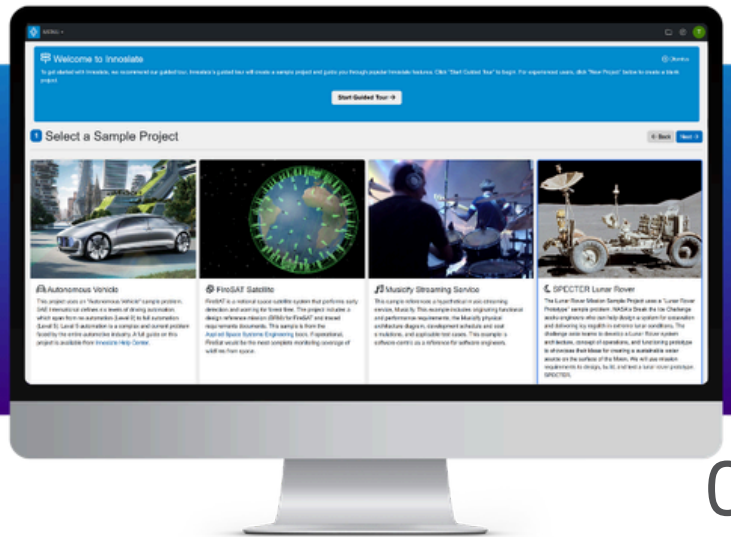


Figure 4. Or with Resource Script User Interface

Figure 5. Or with Resource Script Generated Code

These options will be used to compare with the Resource amount at the time the script is executed. Figure 5 shows the resulting script from this selection. Note that the Global ID is used for getting the Resource from the database (Sim.getAmountByGlobalId). More on this is provided in the Advanced Simulator Scripting section.



LOOP PROBABILITY SCRIPT

In Figure 6, we changed the default probabilities from 50/50 to 80/20 between the Continue and Exit branches. Figure 7 shows the generated script, which allows the simulator to stay in the Continue branch because of the condition, "r <= 80". If the condition is not met, the simulator will leave on the Exit branch.

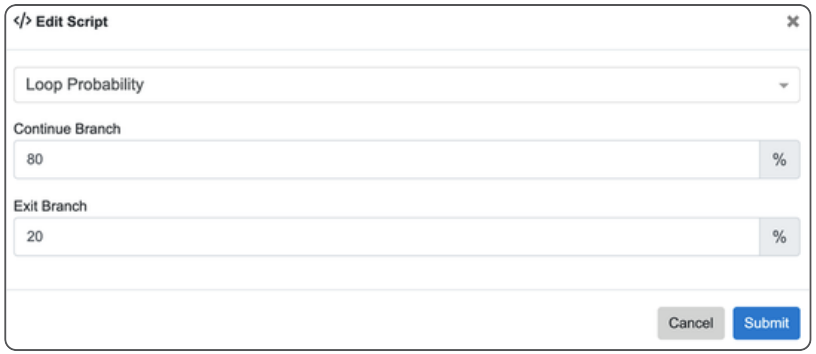


Figure 6. Loop Probability User Interface

Figure 7. Loop Probability Generated Script

LOOP RESOURCE SCRIPT

Figure 8 shows the user interface for the Loop Resource, which looks the same as the OR Resource except condition is based on the Resource's amount value. Figure 9 references the Resource construct in the script by using the REST API, "Sim.getAmountByGlobalId()", and Resource's Global ID.

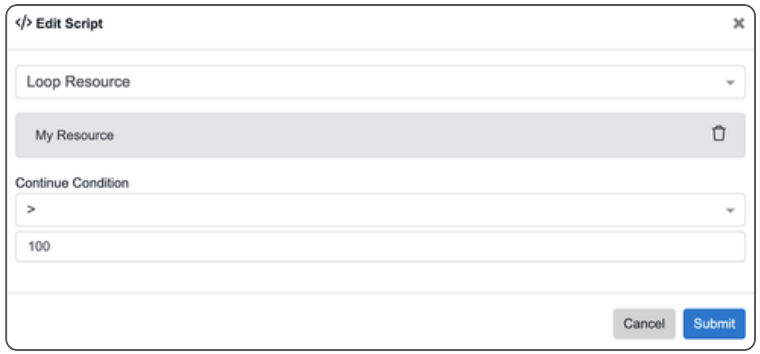


Figure 8. Loop Resource Script User Interface

Figure 9. Loop Resource Generated Script

LOOP ITERATION SCRIPT

Figure 10 shows the Loop Iterations script user interface. The Loop Iteration script allows the user to tell the simulator how many times the loop should run. We changed the number of iterations from the default value of 3 to 10. Figure 11 shows the resulting script. Note that the "i" variable is initialized to zero outside the function, and then "i" is incremented each time this function is called.

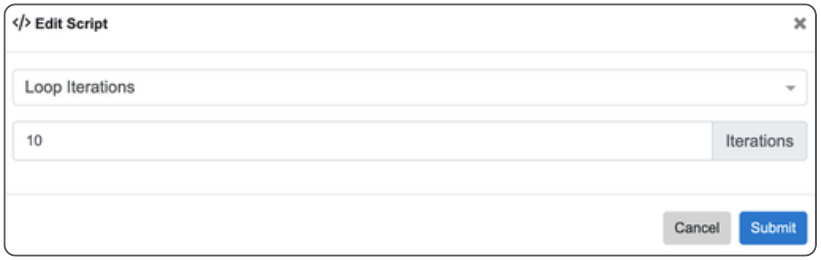


Figure 10. Loop Iterations Script User Interface

We recommend starting with these if you are building a new script, as it will often be easier to get the exit branch identifier automatically. Note that the branch identifier is not essential for the script, and if you want to reuse (copy and paste) the script into other Actions/Decision Points, then you will want to eliminate it.

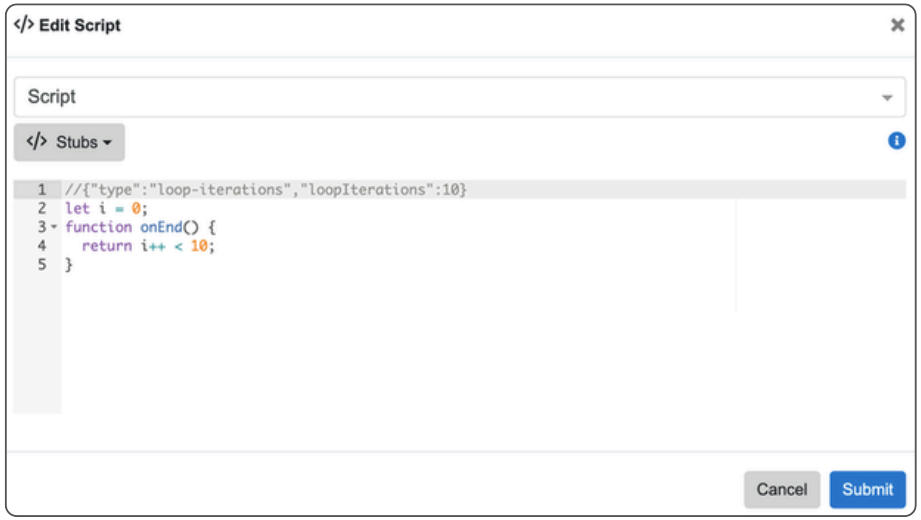


Figure 11. Loop Iterations Generated Script

We also provide "stubs" for those needing to develop scripts from scratch for onStart(), onEnd(), and prompt(). At this point, you will need to know a little JavaScript to write executable programs. Note that prompting the user only works with the discrete event simulator and will throw an error in the Monte Carlo simulator.

One more common situation is the need to make sure that decision points are synchronized with other decision points. The next section discusses this situation.

SYNCHRONIZING OR DECISIONS

Often, we need to make sure that when a decision is made by one Asset, another Asset uses that decision to go down a particular path. We can see this in the example below, Figure 12. *To replicate the examples and experiment hands-on, download the demonstration diagram provided here and upload it to your Innoslate workspace:* Parallel OR Synch Diagram.

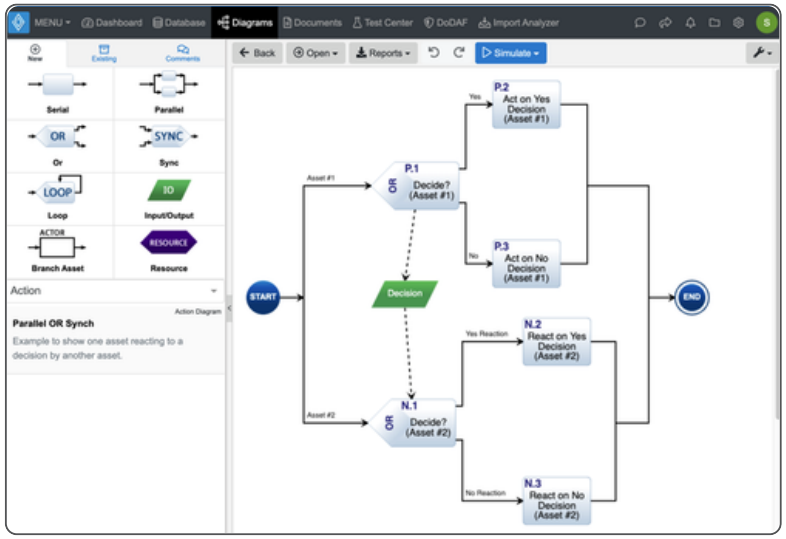


Figure 12. Synchronizing ORs

In this example, the first asset makes a decision using a standard probability script. We want the second asset to respond to that decision and go down a similar path. For this to execute properly, we need to trigger the second decision point from the first, i.e., to ensure that the second decision point does not try to execute at the same time as the first one (which it would do without the "Decision" Input/Output entity.

But we also need to pass on to the second decision point what the decision was (yes or no in this case). This synchronizing of the decision can be accomplished by setting a JavaScript global variable or putting a "Yes" or "No" in the Input/Output attribute field. Let's look at the first one.

Figure 13. Asset #1's Decision Script

Figure 13 shows the first decision point [Decide? (Asset #1)], where it starts with the standard probability script. Then we added the "globals.set" and "print" lines.

For Asset #2's OR script (Figure 14), we get the global variable and use it instead of the probability to make the decision. We also added print statements to show it went down the correct path.

The results for a yes decision are shown in Figure 15. The results for a no decision are shown in Figure 16.

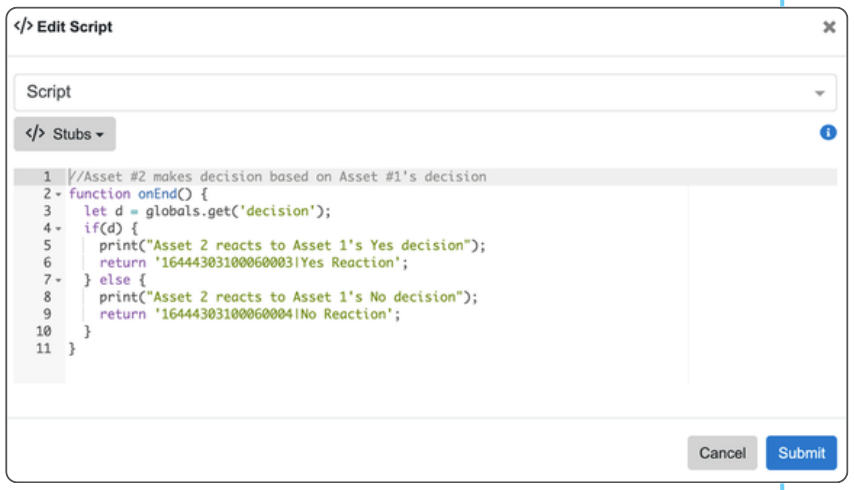


Figure 14. Asset #2's Decision Script

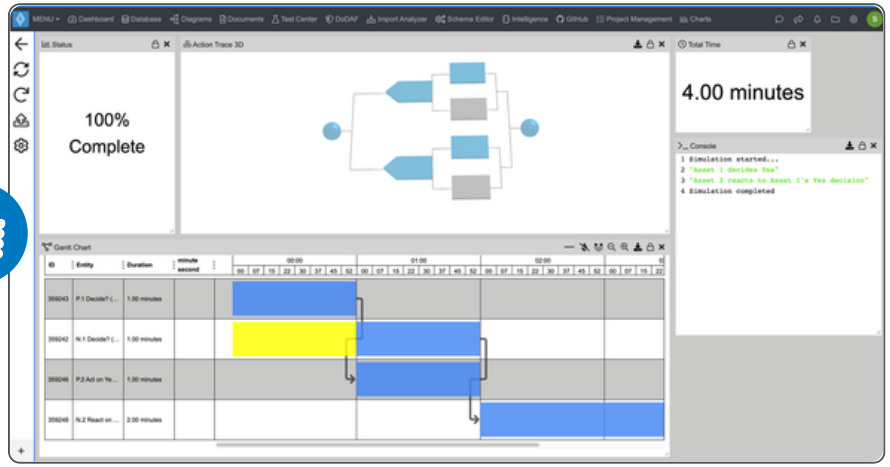


Figure 15. "Yes" Decision Discrete Event Simulator Output

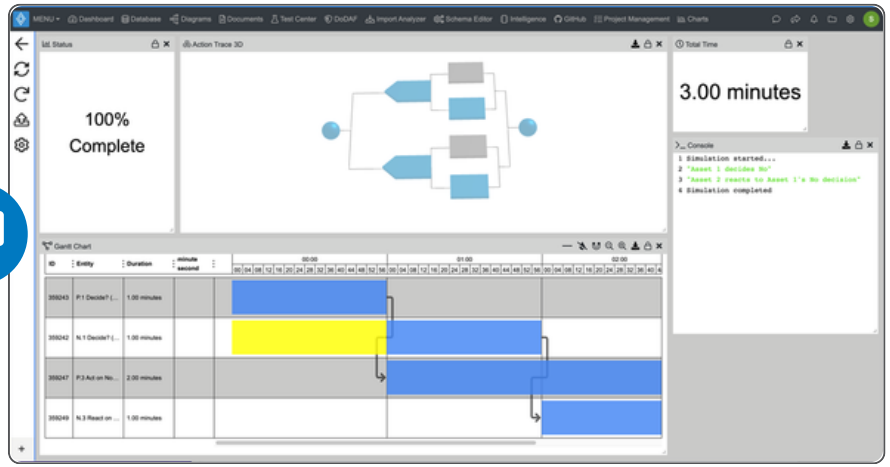


Figure 16. "No" Decision Discrete Event Simulator Output

ADVANCED SIMULATOR SCRIPTING USING “SIM” APIS

The simulator has a scripting API (***Sim***) to allow users to gain access to data contained in the model being executed. To use many of these APIs, you need to provide the Global ID or project-level ID. These IDs can be found when selecting an entity in the Metadata tab on the sidebar.

Sim is a utility class that references methods used to access simulation objects during simulation. Table 1 in the Appendix identifies each of the functions available in the ***Sim*** API and what they return.

Note that Table 1 is missing two methods:

1. *Sim.getResourceByGlobalId(GlobalID)*
2. *Sim.setResourceByGlobalId(GlobalId, amount)*

These objects include:

- Entities (Actions, Resources, I/Os, Conduits, Assets, Cost)
- Current Time
- http Request

You can use these functions to get the names, numbers, descriptions, and any other related attributes of any of the entities available within the simulation. The entities are limited as the simulator only loads information on the entity classes listed above. The example below shows how to use these functions for Cost and Time. Figure 17 shows Actions that incur Costs. The picture shows a simple Action Diagram with two Actions.

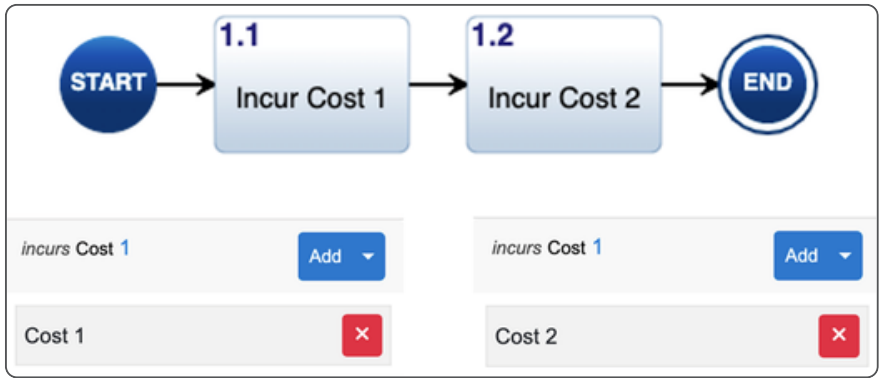


Figure 17. Cost and Time Example

To replicate this example and experiment hands-on, download the demonstration diagram provided here and upload it to your Innoslate workspace: Cost and Time Diagram.

Below the Action Diagram in Figure 17, it shows the *incurs* relationship and Cost entities for each Action. If you add the script in Figure 18 to the Action diagram and then execute it, you will see the results in Figure 19.

Figure 18. Example script

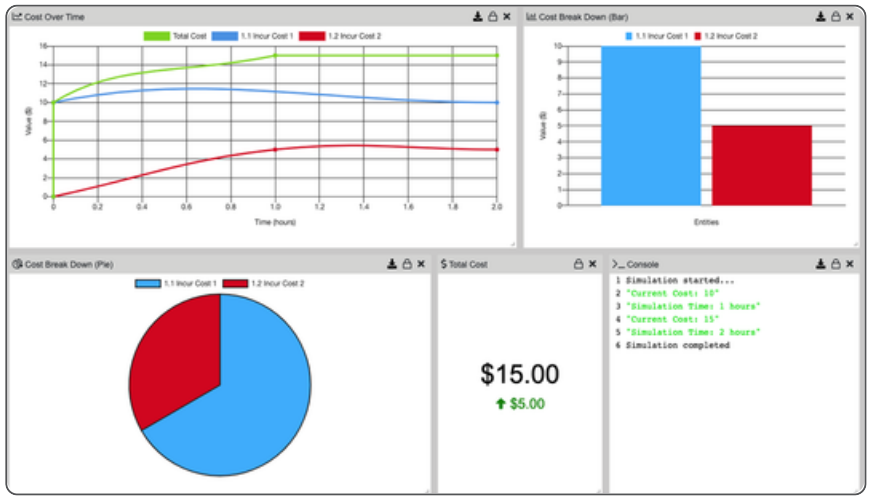


Figure 19. Example Simulation Output

USING RESOURCES IN THE SIMULATORS

Figures 4&5 and 8&9 showed the predefined scripting capabilities in the OR and LOOP actions. These figures show how the pre-built scripts access the current amount of Resources identified by their GlobalIDs. These IDs are found in the Metadata tab of the entity view on the Action Diagram side bar for the Resource, shown in Figure 20.

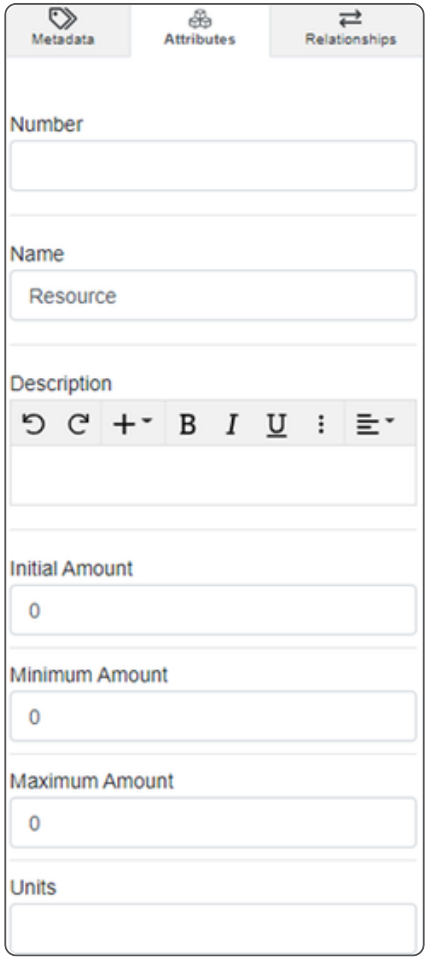


Figure 20. Resource Entity View

The scripts can be manipulated via user-defined scripts of an action and then plotted during the simulation.

Important information about attributes of a Resource: *Value of Resource not being between Minimum and Maximum Amount will cause the simulation to deadlock (break).*

Relationships created between Actions and Resources allow for Pre-Defined Scripting to *produces, consumes,* and *seizes* Resources. Either the *produces* or the *seizes* relationship can be selected when you drag from a green circle on a highlighted Resource onto an Action; an overlay of the two options will be provided. If you select seizes, the Resource icon will turn a lighter purple and the lines will have a "seizes" label.

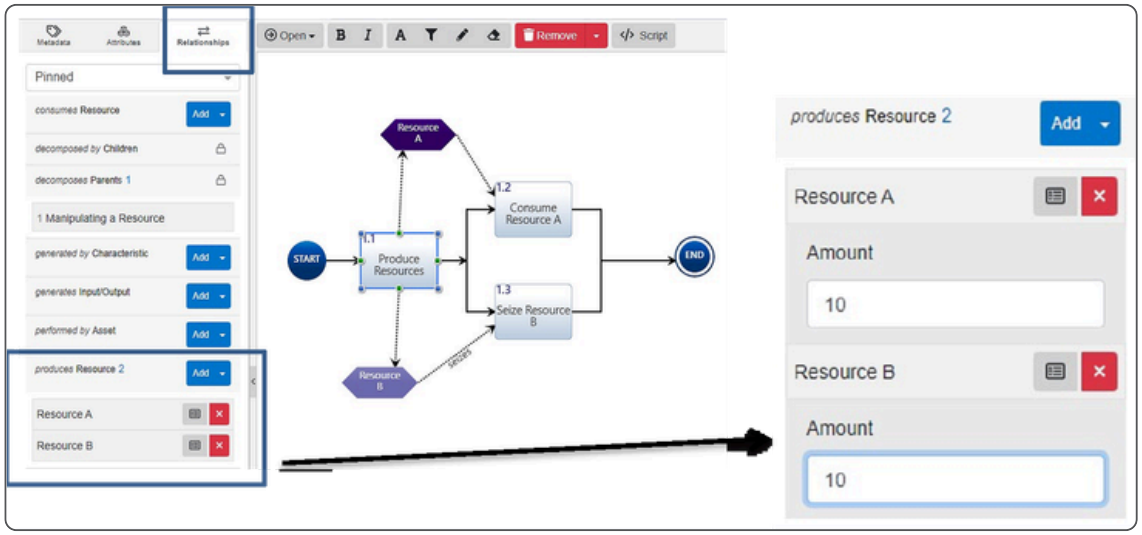


Figure 21. Resource Example

Figure 21 shows an example of the *produces* relationship. Click each Action Entity and go to 'Relationships' on the side panel, find the 'produces' relationship, and define the amount to produce. Figure 22 shows the *consumes* and *seizes* relationships attribute values.

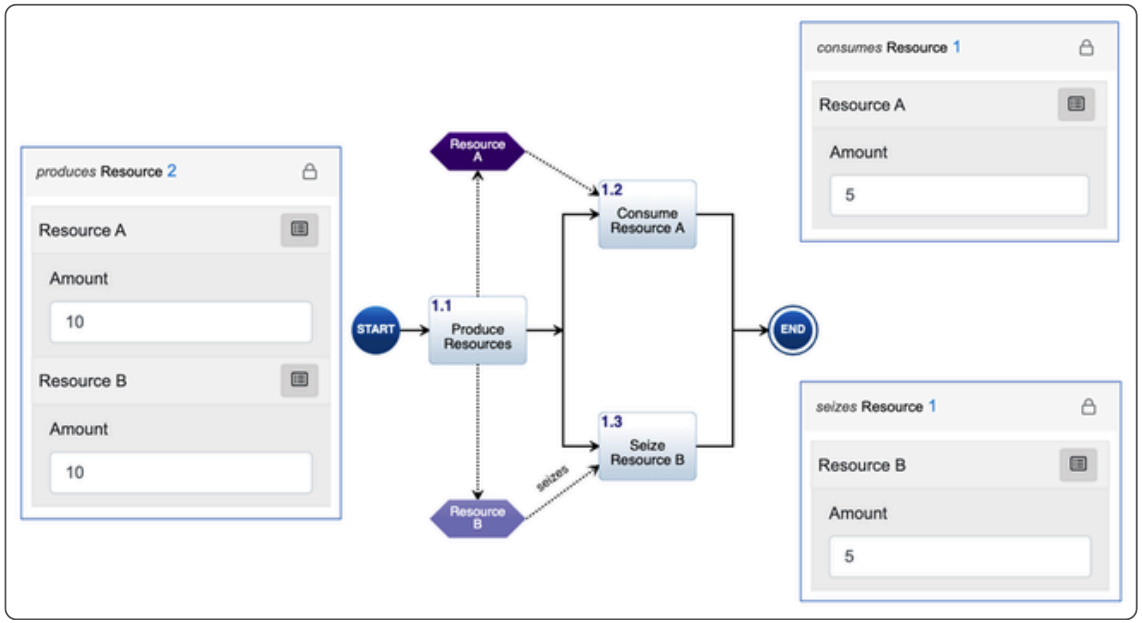


Figure 22. Adding the Seizes and Consumes Relationship

To replicate the example and experiment hands-on, download the demonstration diagram provided here and upload it to your Innoslate workspace: [Resource Diagram Example].

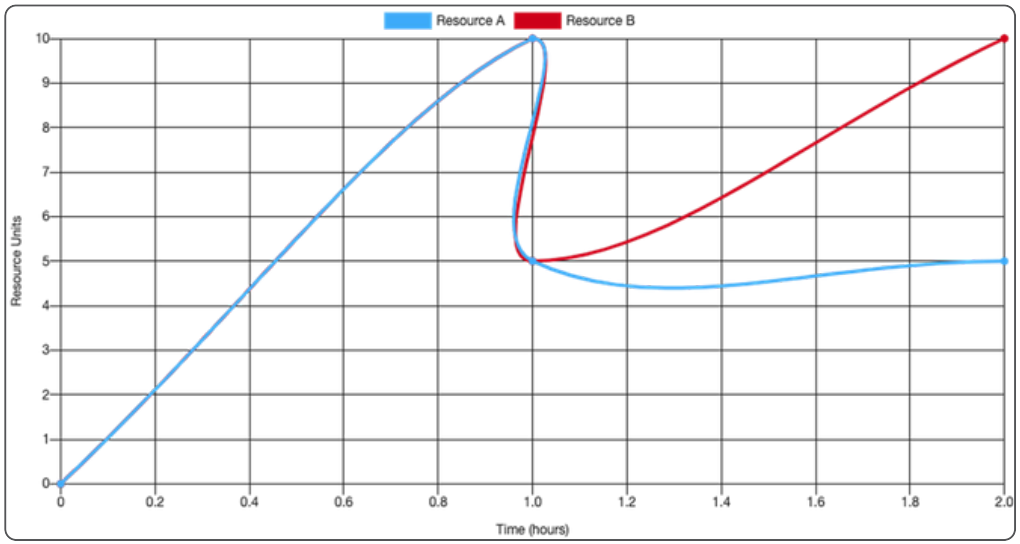


Figure 23. Results of Resource Example

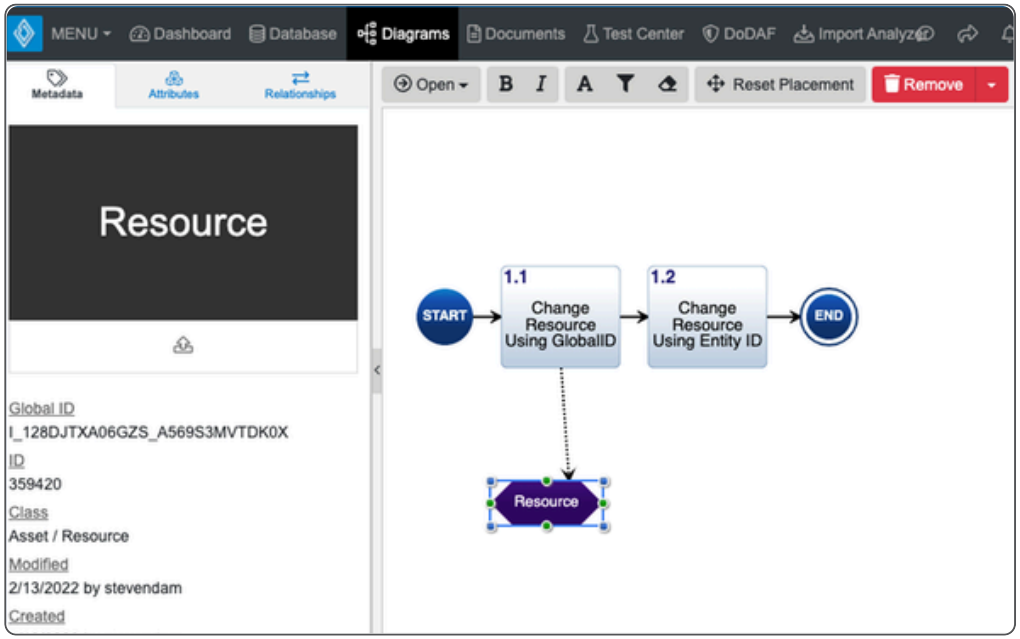


Figure 24. Changing Resource Amounts Using Scripts

Figure 23 shows the results from the simulation. The consumed amount of the Resource is permanently taken away from the Resource, while the seized amount only takes away from the Resource during an Action's duration.

Scripts can also be developed to change resources, with or without the Resource being directly associated with the Action. Figure 24 shows an example of this capability.

To replicate the examples and experiment hands-on, download the demonstration diagram provided here and upload it to your Innoslate workspace: [Resource Manipulation Diagram.]

For this example, we had the Action produce 20 units (remember units are arbitrary and must be consistently set) of the Resource using the attribute on the *produces/produced by* relationship. The scripts for the Actions are shown in Figure 25.

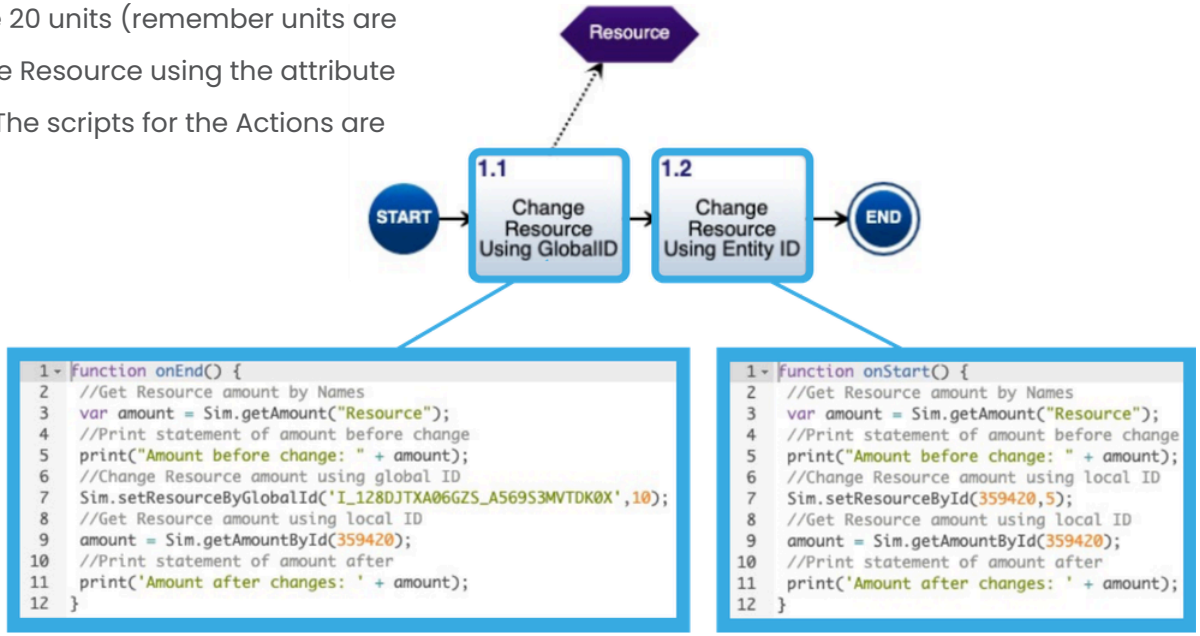


Figure 25. Scripts for Resource Amounts Changed by Script

The resulting simulation output is shown in Figure 26. Note that the *produces* relationship incrementing of the Resource occurs before the execution of the onEnd() script. So the net effect of the first script is to reduce the amount to 10 units from the original 20. The second script reduces the amount of the Resource by another 5 units.

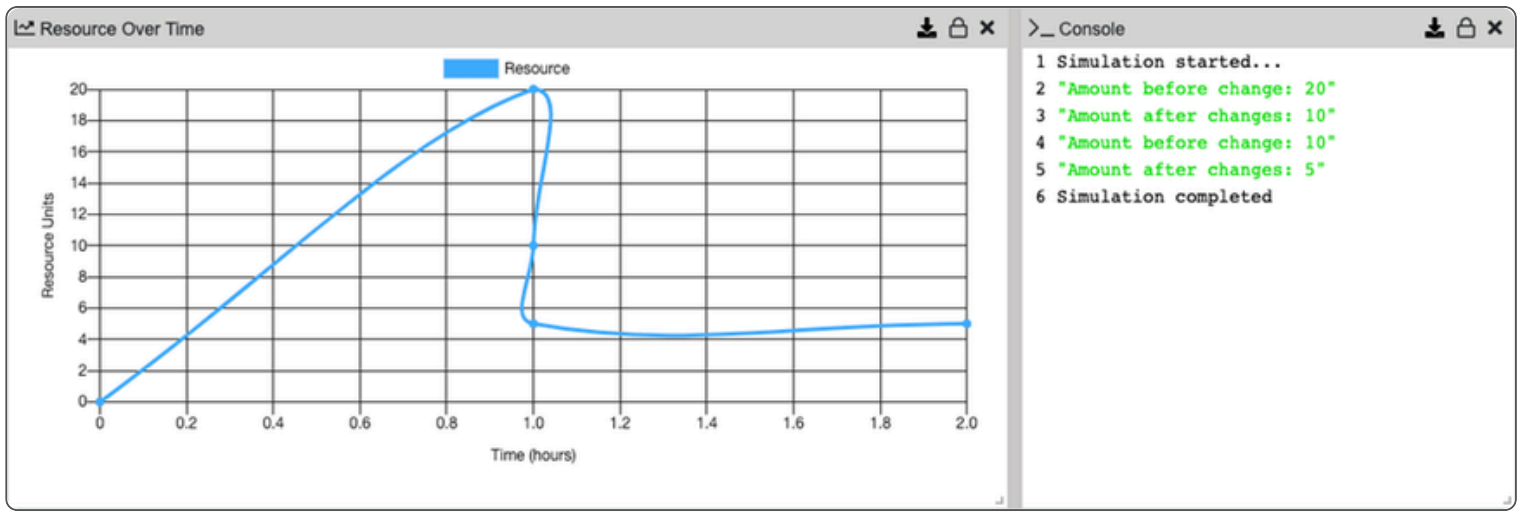


Figure 26. Simulation Output from Changing Resource Amounts by Script Example

USING SCRIPTING PROMPTS

The Innoslate simulator can help conduct tabletop exercises, develop standard operating procedures, and Concepts of Operations. These applications simulate how users may respond to different events and how the system reacts to the user's input. As such, scripts to prompt the user for input have also been provided. Table 2 in the Appendix shows the prompt script methods and their purpose.

Note that these prompts may only be used with the discrete event simulator. If these prompts are enabled during a Monte Carlo simulation, the user will receive an error message. Also note that the "addBooleanInput" method returns a string with the answer, not a Boolean.

An example of using this prompt is shown in Figure 27. The prompt method automatically returns the "response" variable to the script, so that the user only needs to use the onEnd(response) function call.

To replicate the example and experiment hands-on, download the demonstration diagram provided here and upload it to your Innoslate workspace: Resource Prompting Diagram.

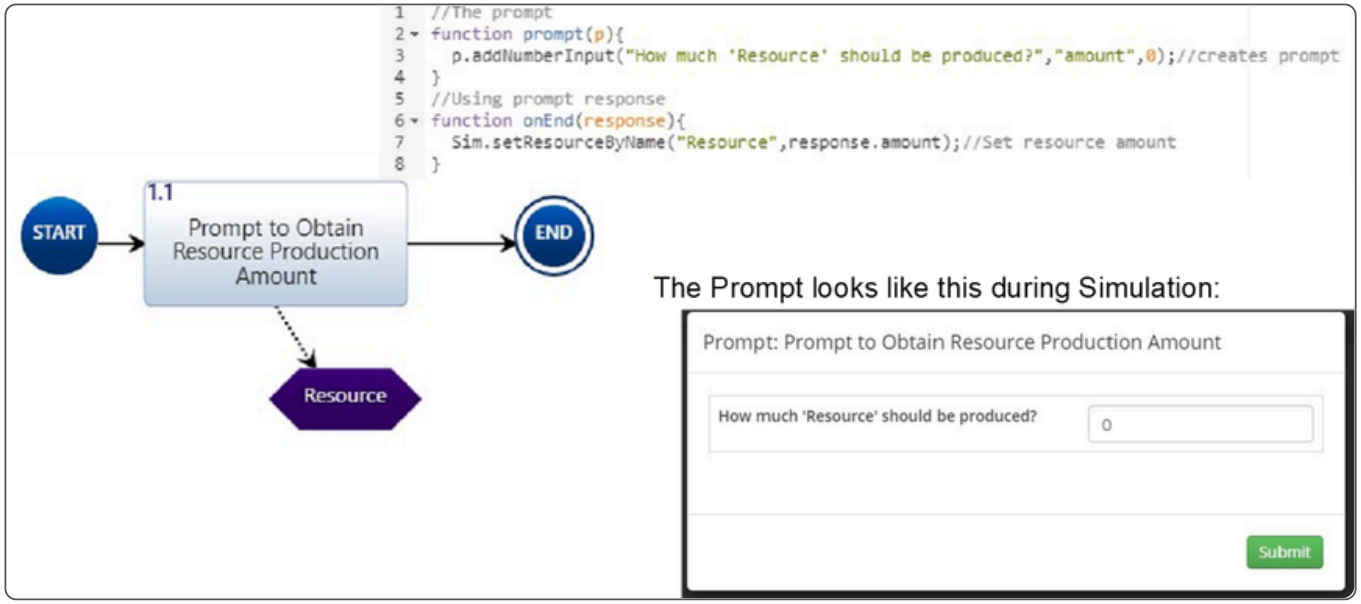


Figure 27. Scripting Prompt Example

ADVANCED SIMULATOR SCRIPTING USING “INNO” APIS

The “Inno” APIs consist of the following methods: InnoObject, InnoEntity, InnoAttributes, InnoSimulation, InnoRelationships, and InnoLabels. These methods can only be used with the entity classes directly related to the instances included in the simulation. They cannot manipulate other classes within the database, such as Risk, Time, Measures, etc. Any changes to the attributes, relationships, and labels are only temporary and contained within the simulation. This limitation is a security feature that makes database corruption impossible. It is also essential for cybersecurity in a cloud environment. To make database changes, you can use the general Java APIs with the Enterprise version of Innoslate. The following subsections discuss each of these methods and their usage.

INNOOBJECT METHODS

InnoObject methods enable the user to get and set most of the common attributes of any object (Name, Description, Created and Modified Dates, etc.). Table 3 in the Appendix shows these methods and their return values. Figure 28 shows an example using some of these methods. Simulation results are shown in the Console panel in the figure.

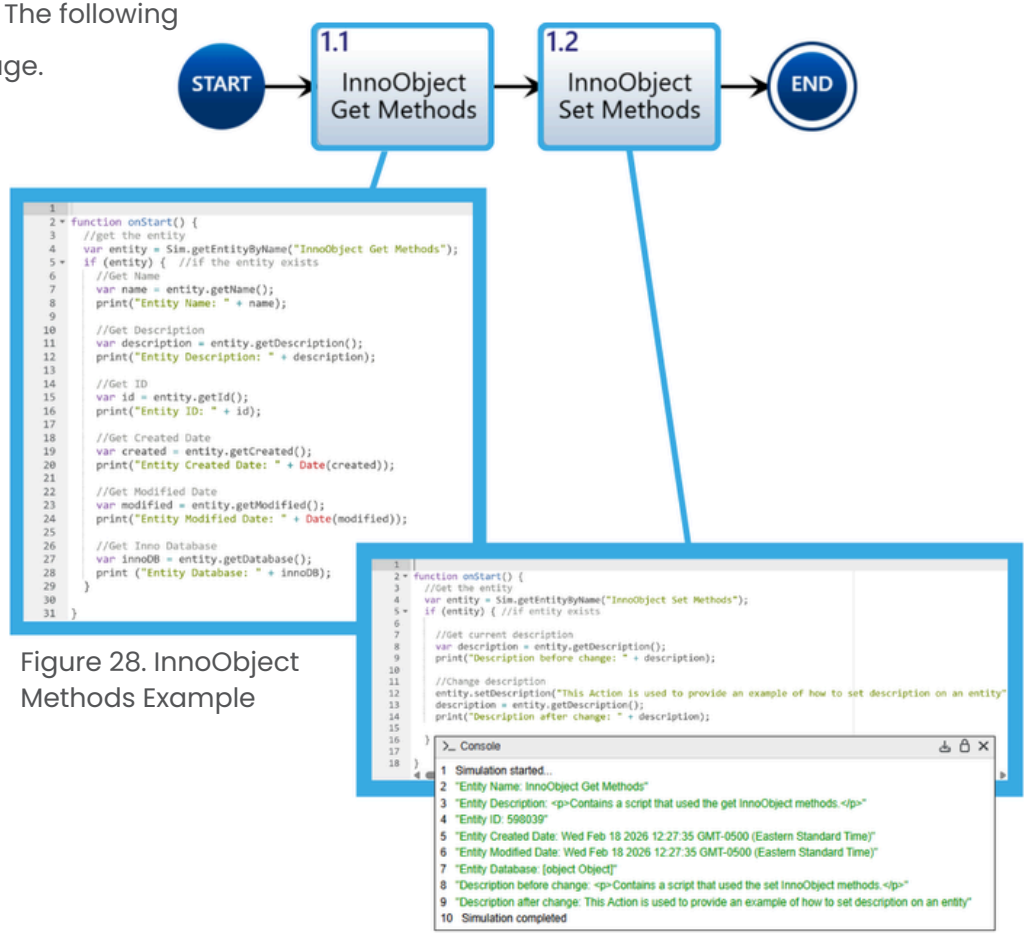


Figure 28. InnoObject Methods Example

To replicate the examples and experiment hands-on, download the demonstration diagram provided here and upload it to your Innoslate workspace: [InnoObject Diagram](#).

INNOENTITY METHODS

InnoEntity methods enable the user to get and set most of the common attributes and properties of any class entity (Number). Table 4 in the Appendix shows these methods and their return values.

An example of using some of these methods is shown in Figure 29. The Console panel shown in this diagram shows the simulation results for the first three actions.

To replicate the examples and experiment hands-on, download the demonstration diagram provided here and upload it to your Innoslate workspace: [InnoEntity Example](#).

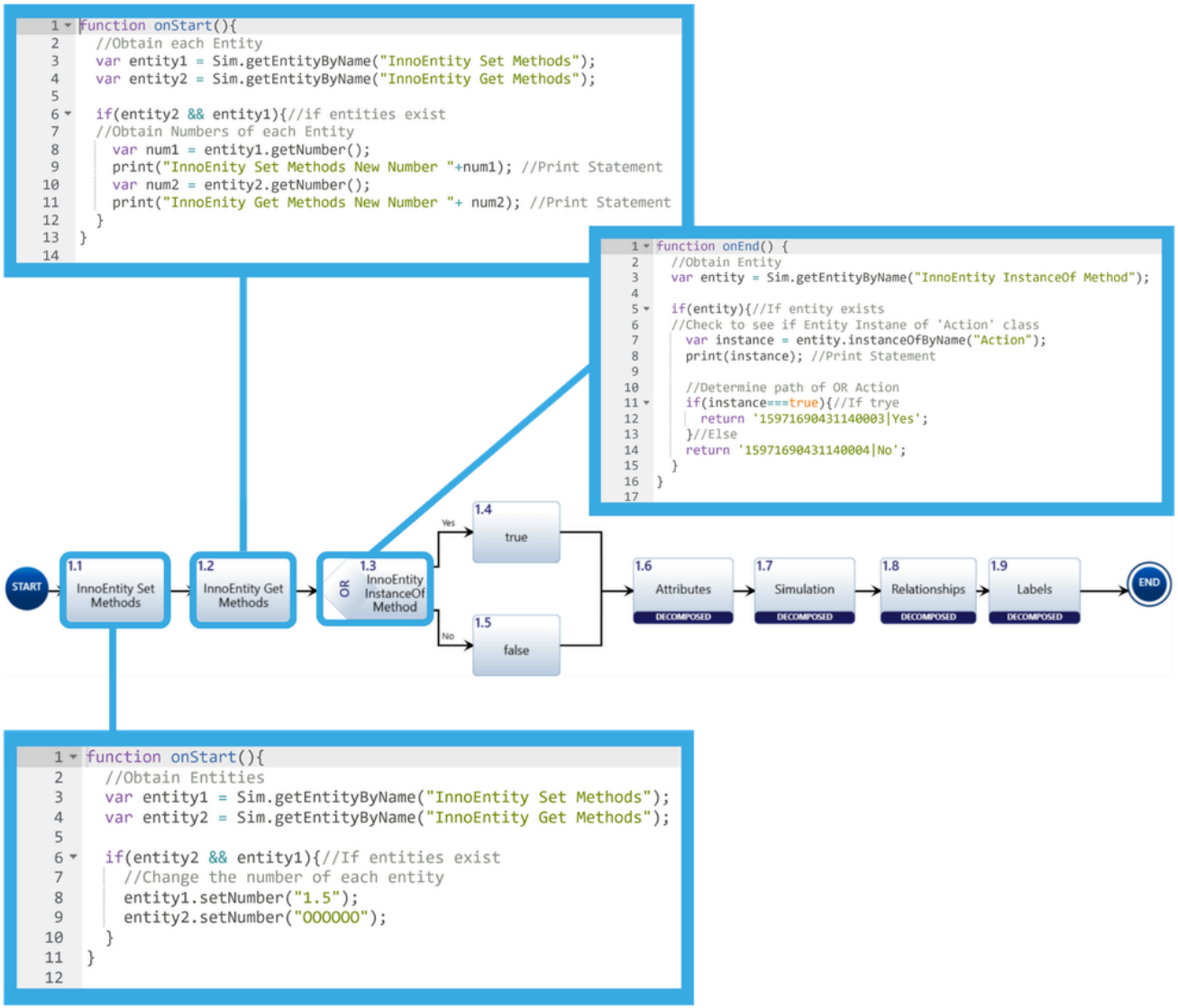
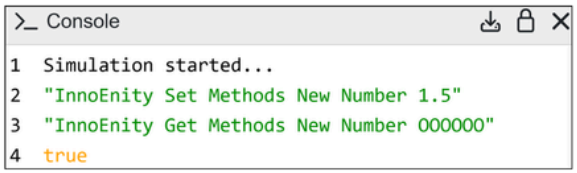


Figure 29. InnoEntity Script Example



INNOATTRIBUTES METHODS

InnoAttributes methods enable the user to get and set any attributes of any object. Table 5 in the Appendix shows these methods and their return values.

An example of using some of these methods is shown in Figure 30. The Console panel shown in this diagram shows the simulation results for these actions.

To replicate the examples and experiment hands-on, download the demonstration diagram provided here and upload it to your Innoslate workspace: [InnoAttributes Diagram](#).

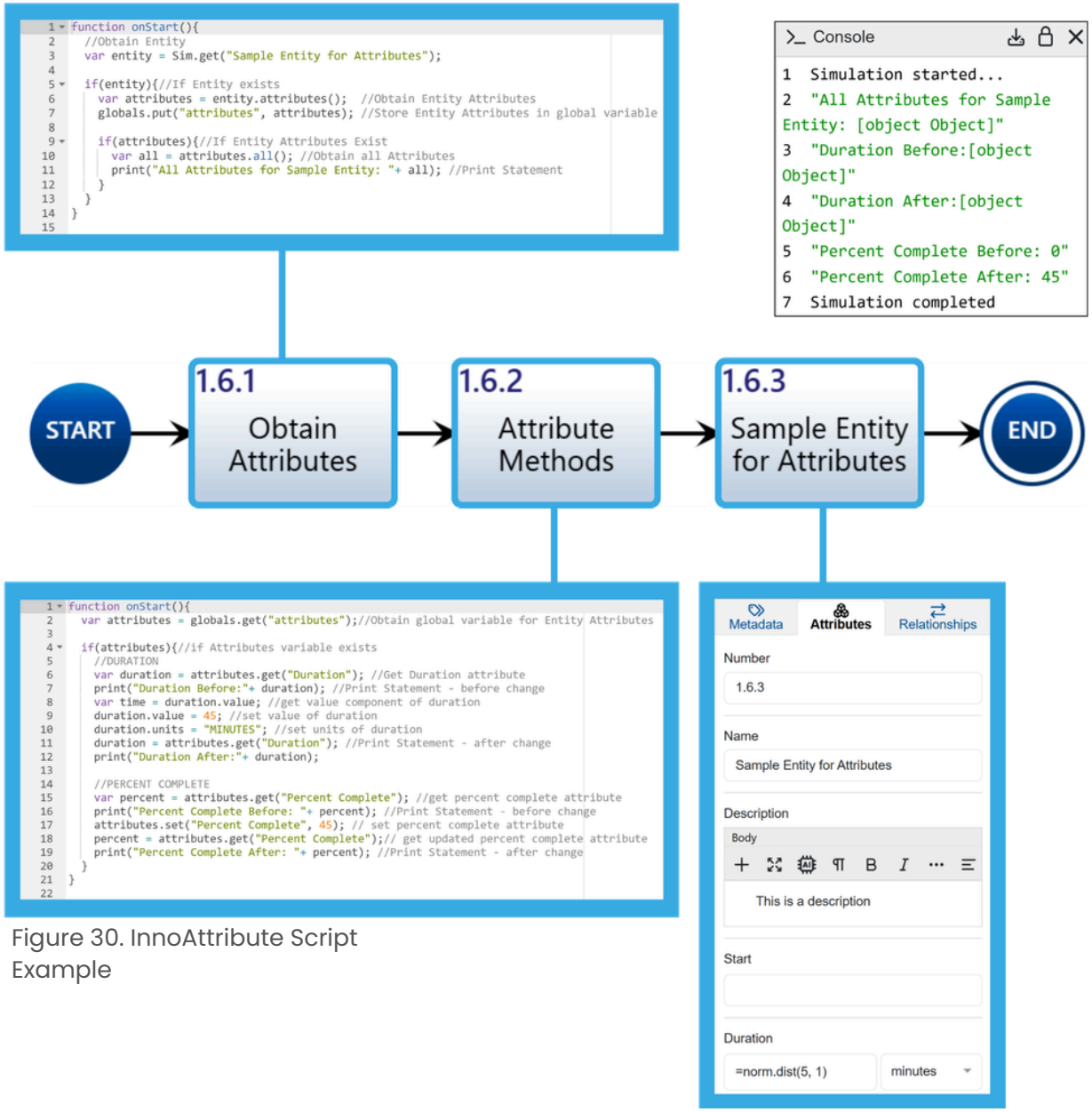


Figure 30. InnoAttribute Script Example

```
function onStart(){
  var entity = Sim.get("Sample Entity for Simulation");//Obtain Entity

  if(entity){//If Entity exists
    var simulation = entity.simulation(); //Obtain simulation part of Entity
    globals.put("simulation", simulation); //Put simulation as global variable
  }
}
```

```
function onStart(){
  //Obtain simulation inforation from global variable
  var simulation = globals.get("simulation");
  if(simulation){//If simulation exists
  //Obtain script from simulation
    var script = simulation.getScript();
    print("Script Before: "+ script);//Print Statement - before change
    var newScript = "//No script"; //New Script to use
    simulation.setScript(newScript); //set script
    script = simulation.getScript(); //obtain updated script
    print("Script After: " + script); //Print Statement - after change
  }
}
```

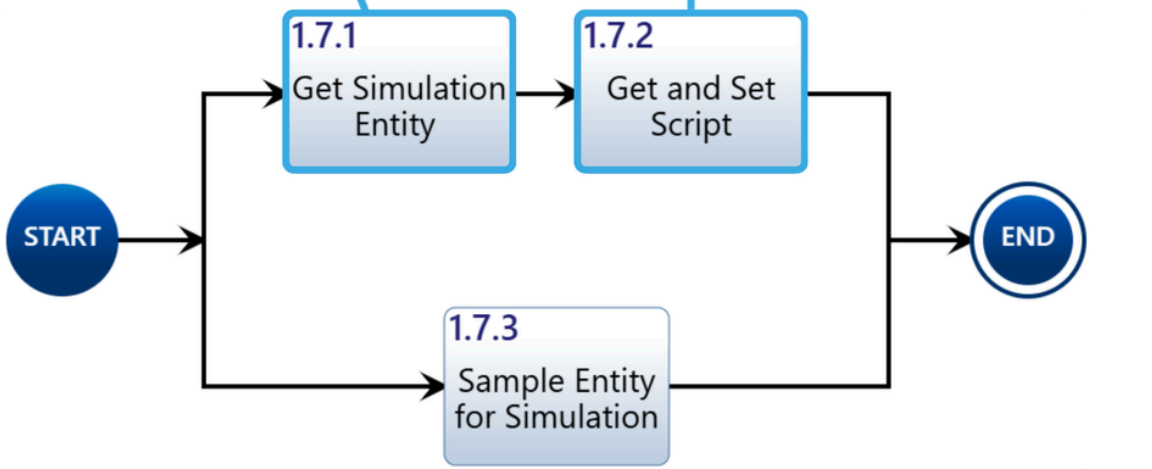


Figure 31.
InnoSimulation Scripts and Simulation Results

```
Console
1  Simulation started...
2  "Script Before: "
3  "Script After: //No script"
4  Simulation completed
```

INNOSIMULATION METHODS

InnoSimulation methods enable the user to get and set simulation scripts. This capability would enable the user to create different scripts for execution within the simulation. Table 6 in the Appendix shows these methods and their return values.

An example of using some of these methods is shown in Figure 31. The Console panel shown in this diagram shows the simulation results for these actions.

To replicate the examples and experiment hands-on, download the demonstration diagram provided here and upload it to your Innoslate workspace: Simulation Diagram.

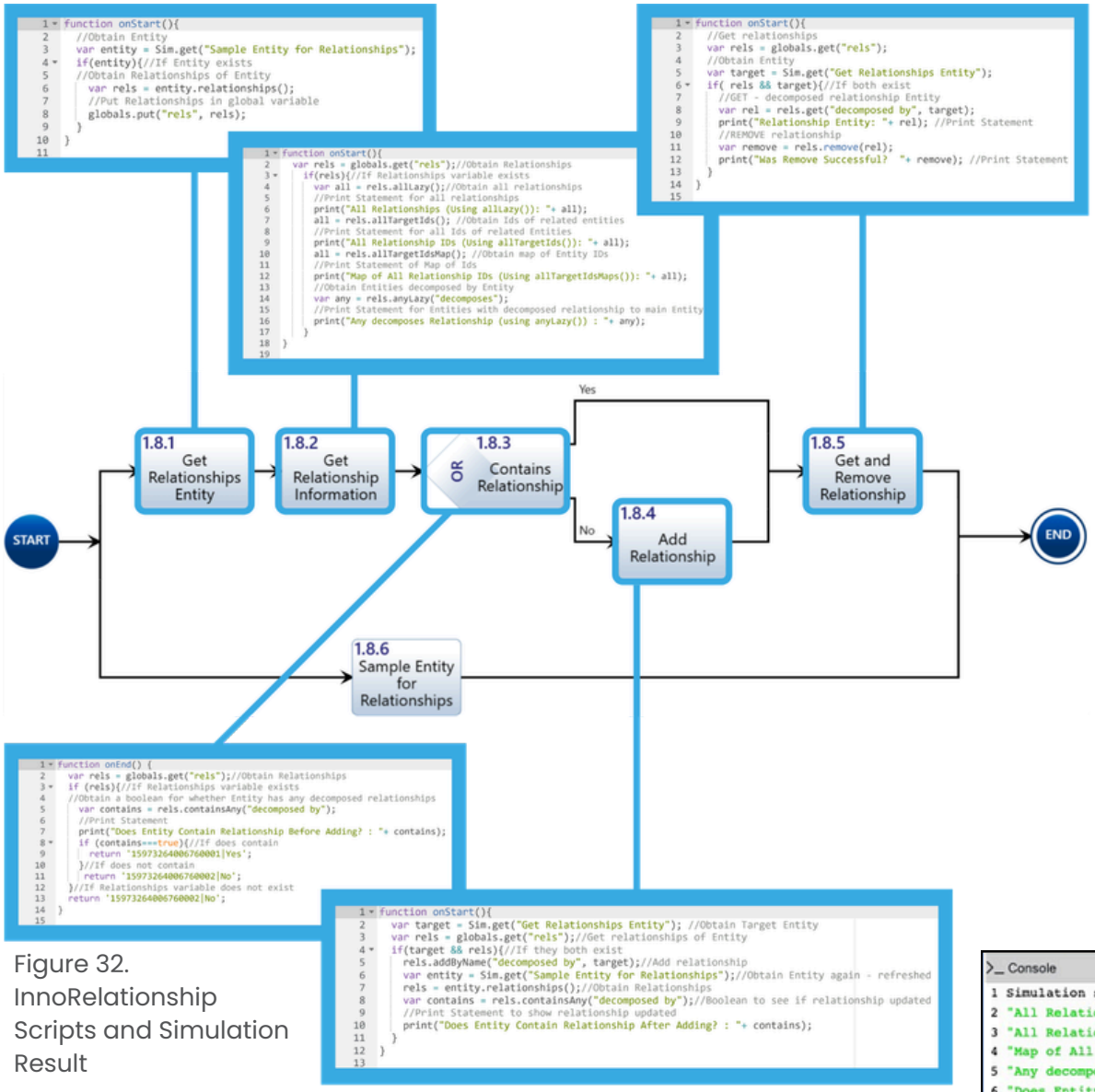


Figure 32. InnoRelationship Scripts and Simulation Result

INNORELATIONSHIP METHODS

InnoRelationship methods enable the user to get and set relationships between entities. Table 7 in the Appendix shows these methods and their return values.

An example of using some of these methods is shown in Figure 32. The Console panel shown in this diagram shows the simulation results for these actions.

To experiment hands-on, download the demonstration diagram provided here and upload it to your Innoslate workspace: [InnoRelationships Diagram].

```
Console
1 Simulation started...
2 "All Relationships (Using allLazy()): [object Object]"
3 "All Relationship IDs (Using allTargetIds()): 64204"
4 "Map of All Relationship IDs (Using allTargetIdsMaps()): [object Map]"
5 "Any decomposes Relationship (using anyLazy()) : [object Object]"
6 "Does Entity Contain Relationship Before Adding? : false"
7 "Does Entity Contain Relationship After Adding? : true"
8 "Relationship Entity: [object Object]"
9 "Was Remove Successful? true"
10 Simulation completed
```

INNOLABEL METHODS

InnoLabel methods enable the user to operations to add, remove, and check for labels. Table 8 in the Appendix shows these methods and their return values.

An example of using some of these methods is shown in Figure 33. The Console panel shown in this diagram shows the simulation results for these actions.

To replicate the examples and experiment hands-on, download the demonstration diagram provided here and upload it to your Innoslate workspace: [Innolabel Diagram](#).

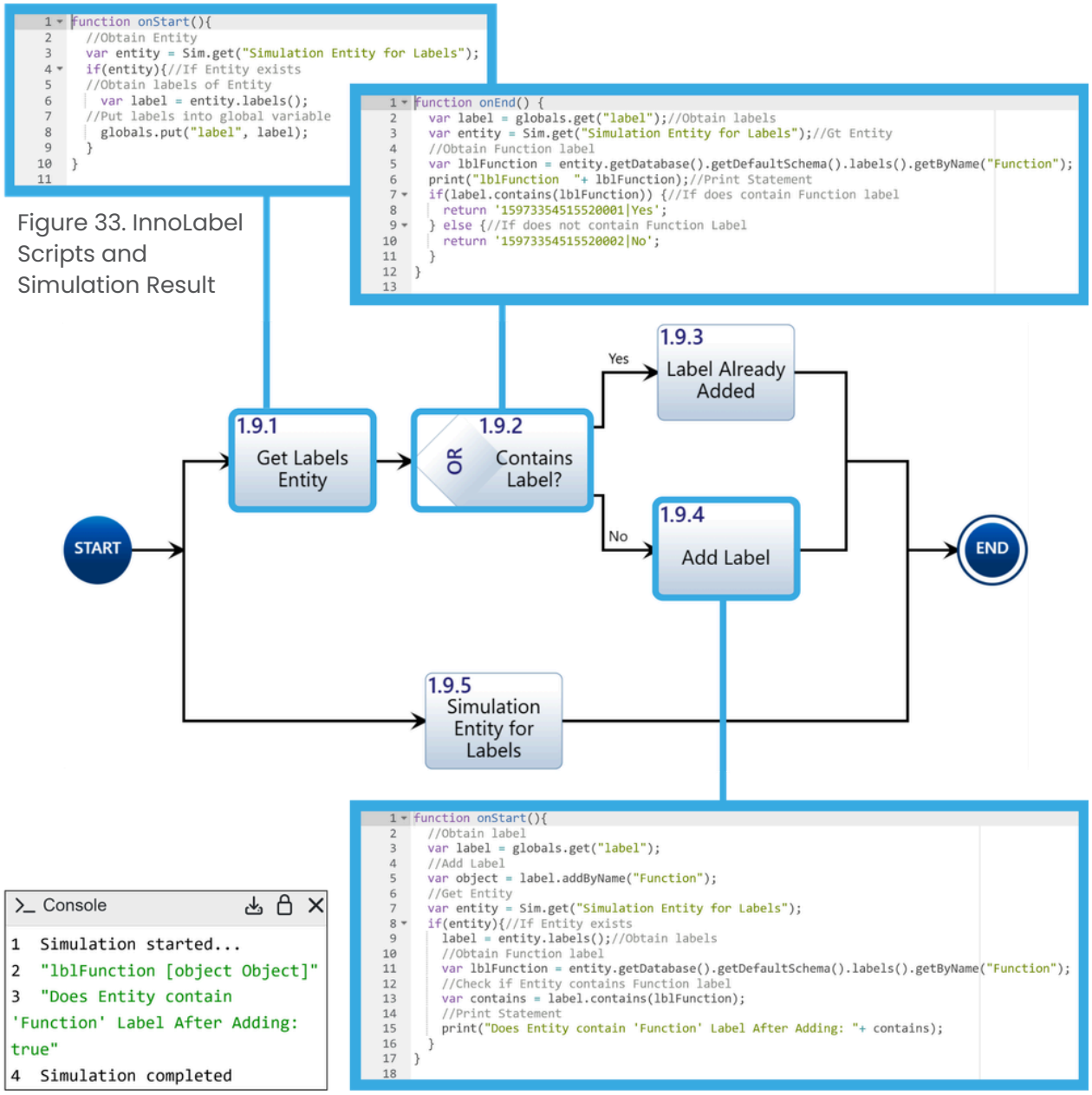


Figure 33. InnoLabel Scripts and Simulation Result

USING I/OS

We have already seen how Input/Output entities are used to control the sequencing of the Actions as a trigger. With version 4.5, a "*Value*" attribute was added to act as a variable in the simulation. This attribute allows information transfer to be more visual in the simulation, instead of just using global variables in the script. The value attribute is a text string and can be encoded with any data type. To access the *Value* attribute, you can use the Sim.getIOValueByNumber(), Sim.getIOValueById(), and Sim.getIOValueByGlobalId() methods for the "Sim" API. To set the I/O value, you will still need to use the InnoEntity methods discussed above. An example is provided below.

USING BASIC I/O API

I/O can be used to store and retrieve values using the I/O construct. The I/O construct uses the following "Sim" API, as shown in Table 9 of the Appendix.

There are two ways to retrieve the I/O value, and it's based on whether the entity has a trigger or not.

If an I/O uses the "generated by" and "received by" relationships, the I/O will be a trigger for the entity with the "received by" relationship.

An example of using some of these methods is shown in Figure 34. The Console panel shown in this diagram shows the simulation results for these actions.

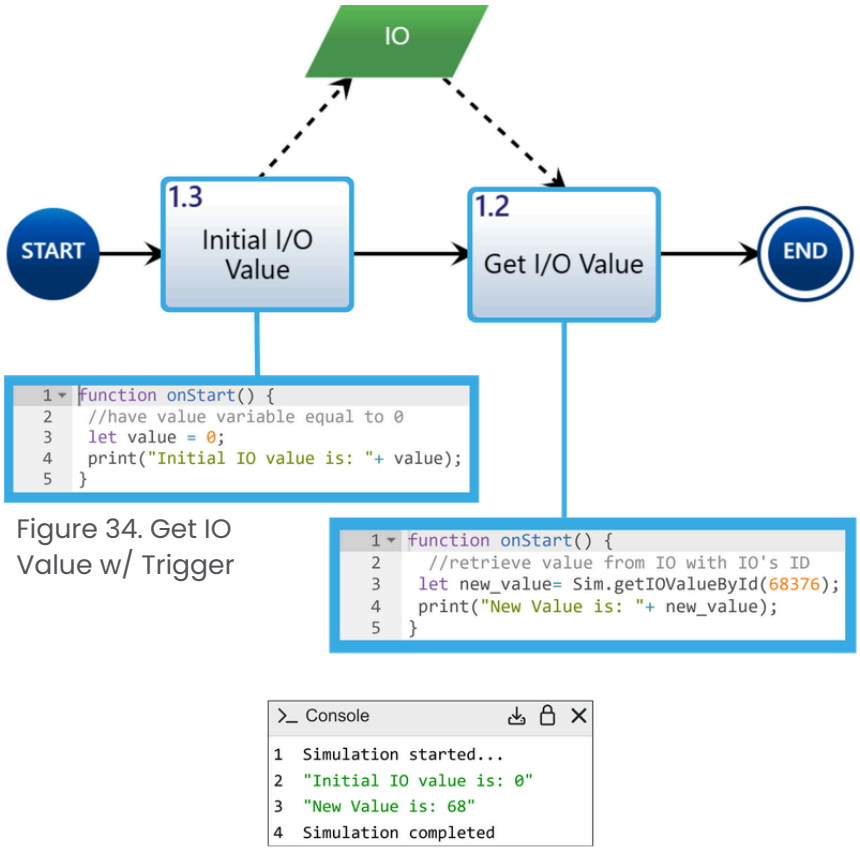


Figure 34. Get IO Value w/ Trigger

If an I/O uses only the “received by” relationship, the I/O will need to be set to 'optional,' and this can be done by going to the I/O "received by" relationship and turning the trigger field into "No" as seen in Figure 35. By doing this, it will avoid a deadlock when the simulation runs.

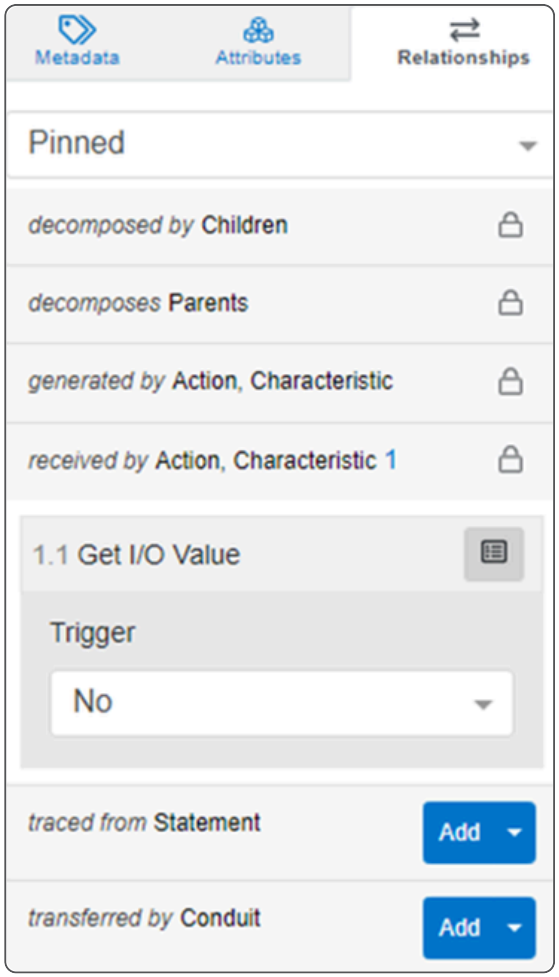


Figure 35. Trigger Field in I/O Entity Relationship

An example of using some of these methods is shown in Figure 36. The Console panel shown in this diagram shows the simulation results for these actions.

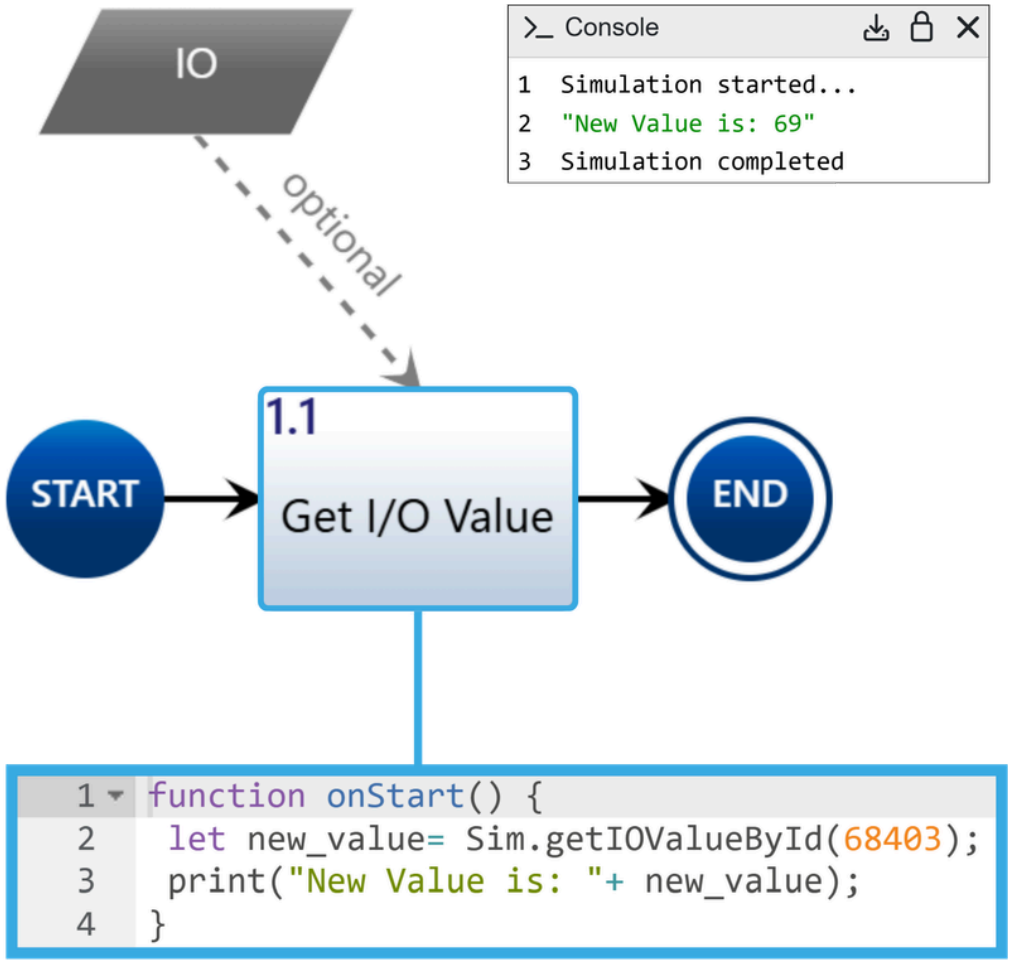


Figure 36. Get I/O Value w/o Trigger

Another use of the I/O is to set a value and retrieve it for later use. To set a value for an IO is to assign a variable using the API and store a new value using "attributes().set()".

An example of using some of these methods is shown in Figure 37. The Console panel shown in this diagram shows the simulation results for these actions.

To replicate the examples and experiment hands-on, download the demonstration diagram provided here and upload it to your Innoslate workspace: I/O Example Diagram.

USING ADVANCED I/O API

The previous section provided the basics for setting and getting the IO entity value. This section will show how a user can set and get IO values from a decomposed parent diagram.

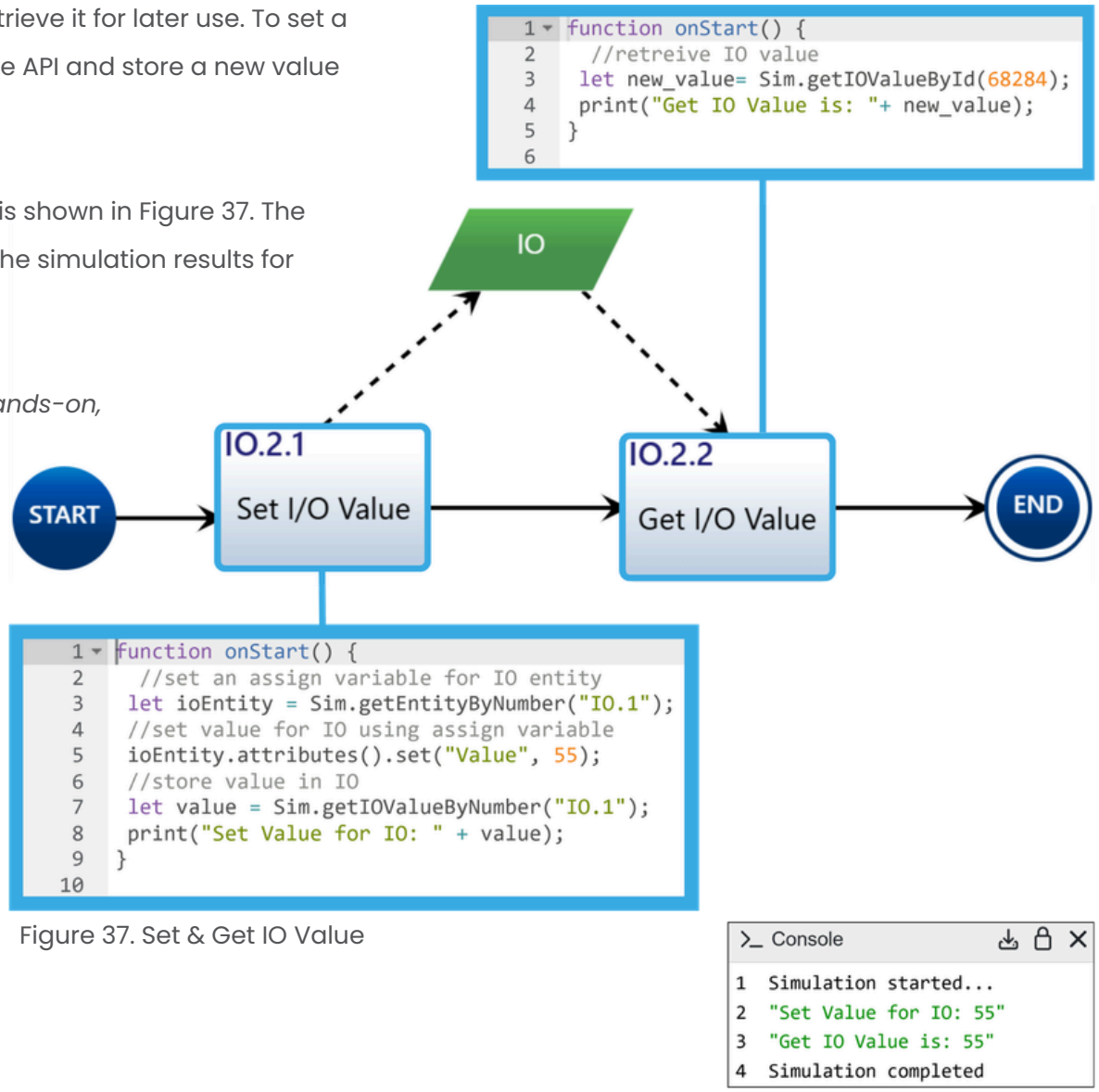


Figure 37. Set & Get IO Value

This is done by making the I/O entity traced to the action entity prior to decomposing that entity. It will ensure the same IO is passed to its child entity.

To set the IO value, the user will need to handle that script in the child entity. An example of using some of these methods is shown in Figure 38. The Console panel shown in this diagram shows the simulation results for these actions.

Once the IO value is set by the child entity, that IO value can be passed and used in the parent diagram. The user will need to make sure the IO is a trigger in the parent diagram to allow the value to be used. An example of using some of these methods is shown in Figure 39. The Console panel shown in this diagram shows the simulation results for these actions.

To replicate the examples and experiment hands-on, download the demonstration diagram provided here and upload it to your Innoslate workspace: [Decomposition I/O Value Transfer Diagram](#).

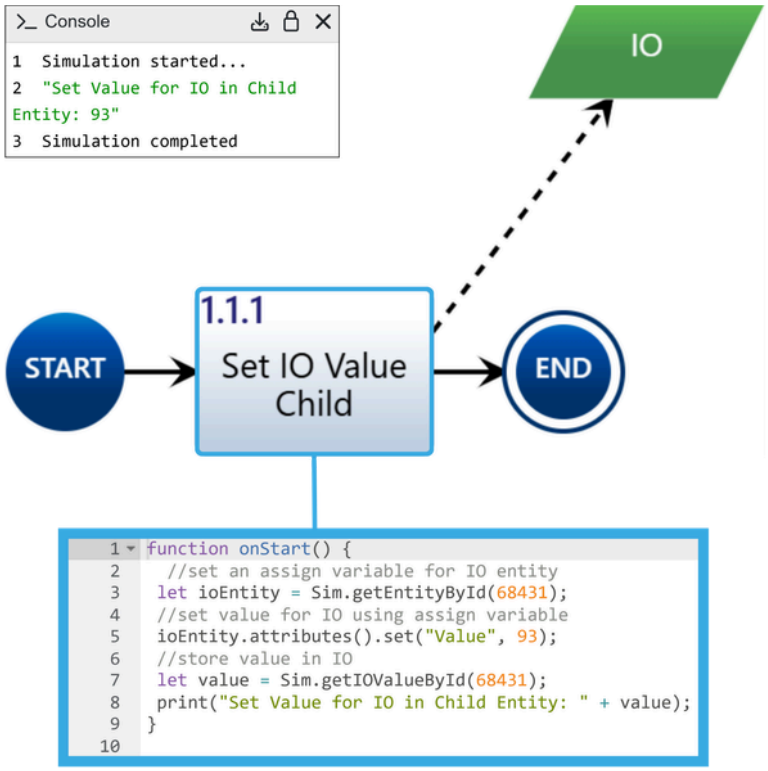


Figure 38. Set I/O Value in Child Entity

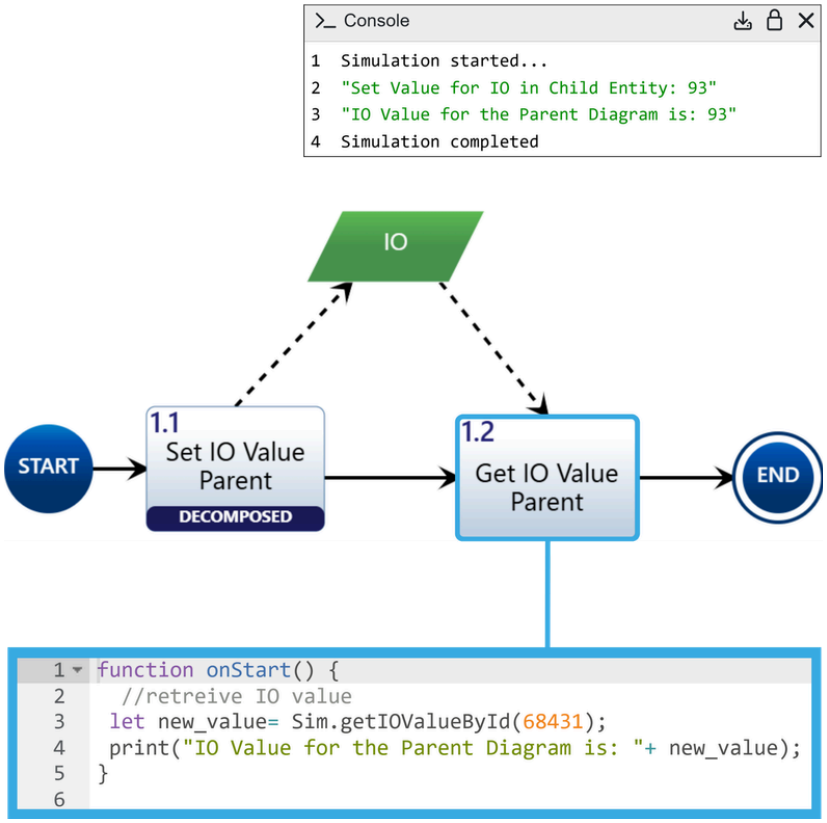


Figure 39. Getting I/O Value for Parent Diagram

CONCLUSION

Innoslate's scripting capabilities enhance simulation flexibility by allowing users to move beyond basic decision logic into more advanced, customized behaviors. While pre-built scripts support many common use cases, JavaScript and the Sim and Inno APIs enable deeper control over data, resources, and system interactions.

By combining these tools with features like synchronized decisions and I/O value handling, users can build more accurate and dynamic simulations. Mastering these capabilities allows for scalable, realistic models that better represent complex systems and operational scenarios.

REFERENCES

[1] Derived from JAVASCRIPT.INFO (https://javascript.info/intro).

[2] See https://javascript.info/coding-style

APPENDIX

Function	Description
Sim.getEntitiesByNumber(number)	Returns a list of entities based on entity's number.
Sim.getEntityByNumber(number)	Returns the first entity result based on entity's number.
Sim.getEntitiesByName(name)	Returns a list of entities based on entity's name.
Sim.get(name)	Short-hand alias for GetEntityByName.
Sim.getEntityById(id)	Returns an entity based on its id.
Sim.setResourceByName(name, amount)	Sets the resource's current amount based on resource's name.
Sim.setResourceById(id, amount)	Sets the resource's current amount based on resource's id.
Sim.getResourceByName(name)	Returns the first resource based on its name.
Sim.getAmount(name)	Returns the resource amount based on resource's name.
Sim.getAmountById(id)	Returns the resource amount based on resource's id.
Sim.getCurrentCost()	Returns the current cost of the simulation.
Sim.getIOValueByNumber()	Returns the Input/Output value attribute based on entity number.
Sim.getIOValueById()	Returns the Input/Output value attribute based on entity id.
Sim.getIOValueByGlobalId()	Returns the Input/Output value attribute based on entity global id.
Sim.getCurrentTime()	Returns the current time of the simulation.
Sim.httpGet(url)	Allows users to do a synchronous GET request, which will get data from an external server.
Sim.httpPost(url)	Allows users to make a synchronous POST request that creates data on an external server.
Sim.httpPut(url)	Allows users to do a synchronous PUT request that updates data on an external server.

Table 1. Simulator Scripting Functions

Function	Description
addEnumInput(title, name, choices, value)	Prompts user to choose from set of predefined choices.
addTextInput(title, name, value)	Prompts user to enter text.
addBigtitleTextInput(title, name, value)	Prompts user to enter a large amount of text.
addNumberInput(title, name, value)	Prompts user to enter a number.
addBooleanInput(title, name, value)	Prompts user to choose from True or False.
setTitle(title)	Sets the title of the prompt.

Table 2. Scripting Prompt Methods

Method	Return
getCreated()	Creation time in milliseconds
getCreatedBy()	User that created the object
getDatabase()	Database for current entity
getDescription()	Description text of object
getDescriptionAsText()	Strips out special characters & formatting
getGlobalID()	Global ID of the entity
getId()	ID of the entity
getModified()	Modified time in milliseconds
getModifiedBy()	User that last modified the object
getName()	Name of the object
isHidden()	Determines if object is a hidden object
IsLocked()	Determines if object is locked
setDescription(text)	Updated entity description
setHidden(hidden)	Sets object as hidden
setLocked(locked)	Sets the locked state of object
setName(name)	Updated entity name

Table 3. InnoObject Methods

Method(s)	Return
setInnoClass(name), setInnoClassByName(name)	Entity Object
setInnoClassById(id)	Entity Object
getInnoClassId()	Id number
getNumber()	Entity's number
hasChanged()	Boolean: true || false
instanceOf(name), instanceOfByName(name)	Boolean: true || false
instanceOfById(id)	
setNumber(number)	Entity

Table 4. InnoEntity Methods

Method(s)	Input	Return
all()		*id: attribute value* of all attributes
get(name), getByName(name)	String	*id: attribute value*
getById(id)	Integer	*id: attribute value*
getByProperty(property)	InnoProperty	*id: attribute value*
set(name,value), setByName(name,value)	String, String	InnoEntity
setById(id,value)	Integer, String	InnoEntity
setByProperty(property, value)	InnoProperty, String	

Table 5. InnoAttribute Methods

Method	Input	Return
getScript()		String of the script
setScript(script)	String	

Table 6. InnoSimulation Methods

Method(s)	Return
add(relationName, target), addByName(relationName, target)	Entity
addById(relationId, target)	Entity
addByRelation(relation, target)	
allLazy()	Array of all InnoEntity related to Entity
allTargetIds()	all IDs of related Entities
allTargetIdsMap()	Map of IDs
allTargets()	Array of InnoEntity objects
get(name,target), getByName(name, target)	EntityRelationships object if exists, or false
getById(id, target)	EntityRelationships object if exists, or false
getAndRemove(name,target)	EntityRelationships object if exists, or false
remove(relationship)	True if successful
anyByIdLazy(id)	Array of InnoRelations
anyLazy(name), anyByNameLazy(name)	Array of InnoRelations
anyTargetIds(name)	IDs of related Entities
anyTargetIdsById(relationId)	IDs that have given relationship with
contains(relationship)	Boolean
containsAny(relationName), containsAnyByName(relationName)	Boolean
containsAnyById(id)	Boolean
containsById(relationId, target)	Boolean
containsByName(relationName, target)	Boolean
countAny(relationName), countAnyByName(relationName)	Integer
countAnyById(id)	Integer

Table 7. InnoRelationship Methods

Method	Input	Return
add(label)	InnoLabel	InnoEntity
addById(id)	Integer	InnoEntity
addByName(name)	String	InnoEntity
all()		*label id:InnoLabel* pairs
contains(label)	InnoLabel	Boolean
containsById(id)	Integer	Boolean
containsByName(name)	String	Boolean
ids()		array of label IDs
remove(InnoLabel)	InnoLabel	

Table 8. InnoLabel Methods

Method	Input Value	Input Type	Return
Sim.getIOValueByNumber()	Entity's Number Attribute	String	Entity Attribute Value
Sim.getIOValueById()	Entity's ID	Integer	Entity Attribute Value
Sim.getIOValueBYGlobalId()	Entity's Global ID	Integer	Entity Attribute Value

Table 9. I/O Sim API