

Prerequisites of a model to be RL-ready for the Bonsai platform

The focus of this document is on the process of adding the required dependencies into an AnyLogic model to properly connect it with the Bonsai platform, enabling the use of an AnyLogic model as the simulator for training in the reinforcement learning loop. It's assumed that the intended model to use with the Bonsai platform is RL-ready. The strategic process behind best refactoring or building a model that is RL-ready is not covered in this document.

To put it briefly, a simulation model that is used for reinforcement learning delegates some of the decisions (actions) that are being taken throughout its execution to the learning agent (in this case, the Bonsai brain).

Therefore, the RL-ready model should be able to:

- Set a desired configuration as the initial state of each [training] episode
- Pause itself at the moments that the delegated decisions should be taken (episode step)
- Communicate its current state to the brain (observations)
- Implement the chosen action by the brain in the model

There are two types of decision points (episode steps) you can configure:

- 1) Decisions that are made in pre-defined time intervals (e.g., every 6 hours)
- 2) Decisions that are made at specific events in the model (e.g., call-back fields of process blocks, condition-based events, transitions of statechart)

As mentioned, there is a lot more to designing/refactoring simulation models that are RL-ready. This brief introduction served to covers some of the fundamental ideas. The following section goes into the steps needed to convert your RL-ready model to a simulator in Bonsai platform.

Using the “wrapper” model and the Bonsai Connector library to prepare your RL-ready model to be a Bonsai simulator

“Wrapping” is the process of incorporating your RL-ready model into a second model named “Wrapper Model”. This second model already includes all the needed dependencies to make the proper connection to the Bonsai platform. To start the wrapping process, there are two necessary assets that are available for download from the AnyLogic website on the [bonsai webpage](#).

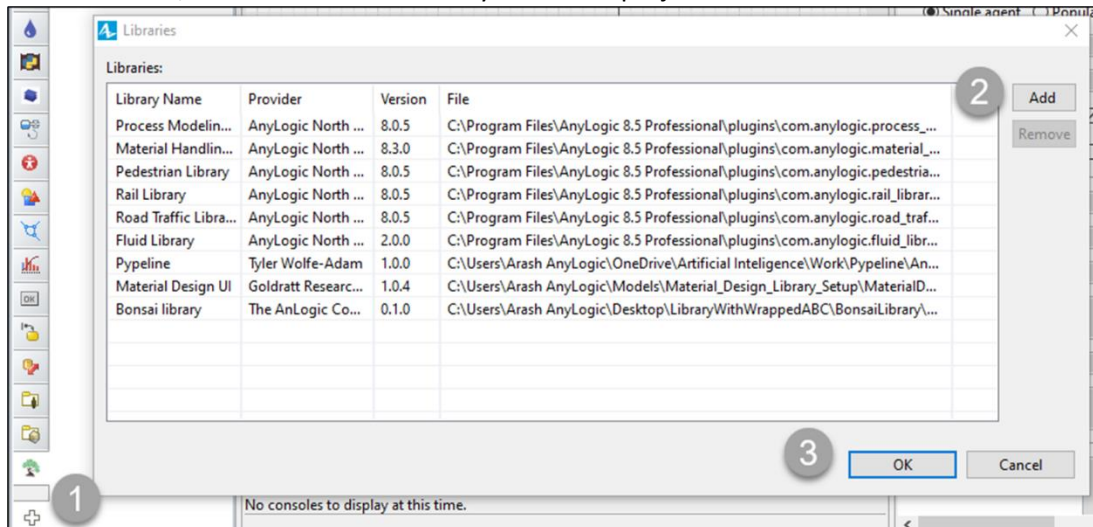
- I. Bonsai Connector Library: This is a custom AnyLogic library that, under the hood, is used to communicate between your model and the Bonsai platform.
- II. Wrapper Model: This is similar to any other AnyLogic model, but it is not intended to be used by itself. Instead, your RL-ready model will get incorporated into it and is what you will use as the simulator. By using the model, the setup process is simplified, as it comes with the required components that you simply need to fill out.

Note: If you are just experimenting with any of the two already wrapped examples ([Activity Based Costing Analysis](#) or [Product Delivery](#)), you just need to finish step 1 to add the Bonsai Library to the AnyLogic. These models have the wrapper model directly integrated into them and are ready to be used as simulators in Bonsai. This was done to simplify the distribution of files, however directly incorporating the wrapper model is not the recommended approach.

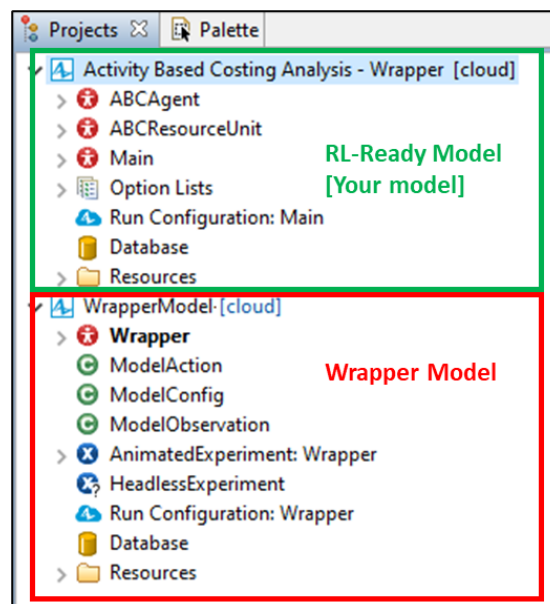
To wrap your RL-ready model and make it ready to be used in the Bonsai platform:

1. Add the Bonsai Library to the AnyLogic.
 - a. Download the library from the provided link and save it to a location on your hard drive where it won't be moved
 - b. Inside AnyLogic, click on the small plus sign on the lower left corner the Palette panel > Manage Libraries... > Add
 - c. Browse to where the library jar file is located on your hard drive, then confirm all the open windows.

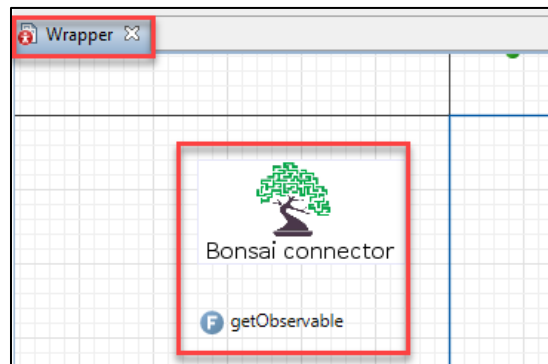
Note: You only need to add this library once to AnyLogic. As long as the jar file is not moved or deleted, it will be available to all your future projects.



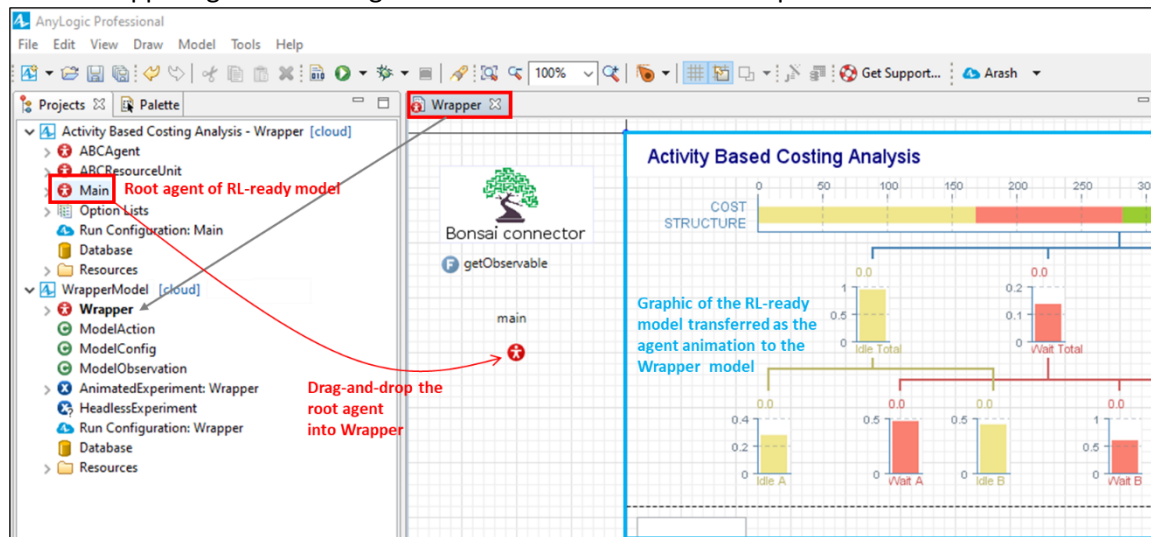
2. Open the wrapper model alongside the RL-ready model. The ABCA model shown below is just an example to showcase the process.



- Open the Wrapper agent type from the wrapper model. In the editor panel, if you pan to the left, you should see the Bonsai connector object (which comes from the library you added in step 1) and a getObservable function.

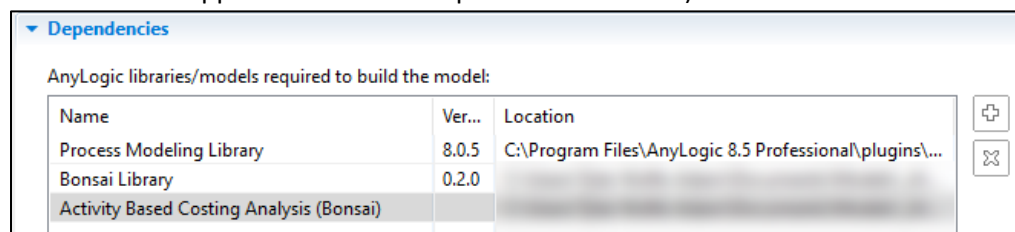


- Drag and drop the root / top-level agent of your model (e.g., Main) from the project panel into the Wrapper agent. The image below shows a visual of this step.



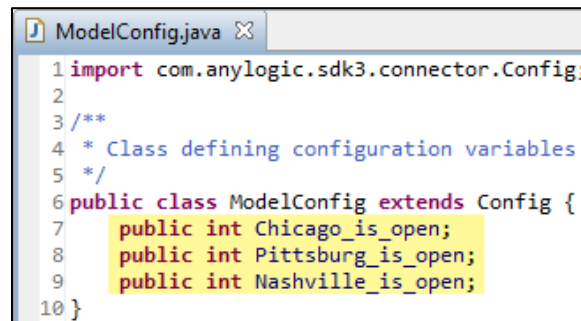
While agent types are typically instantiated from within one model, in this case it's being done across separate models. When you do this, all the animation from your model's root agent will appear inside the Wrapper agent.

In addition, your model is introduced as a dependency to the wrapper model (you can see this in the Properties window of the wrapper model under Dependencies section):



This step incorporates your original model into the wrapper; from this point on, the wrapper model is in control of your Main agent type.

The next part of the process is to define your model's configuration, its action space, and its observation space. For these, note that there are three Java classes included with the wrapper model: ModelConfig, ModelAction, and ModelObservation. Each one will require you to add Java variables representing the type and name of each attribute you want to include. For an example, see the declaration of the ModelConfig variables used in the Activity Based Costing Analysis example model, highlighted in yellow below:



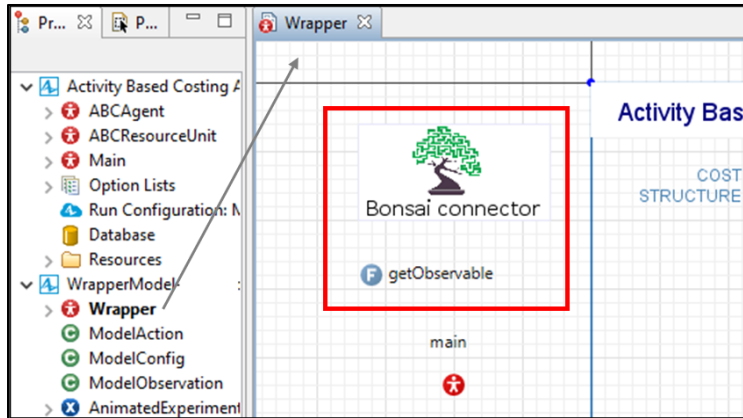
```
1 import com.anylogic.sdk3.connector.Config;
2
3 /**
4  * Class defining configuration variables
5  */
6 public class ModelConfig extends Config {
7     public int Chicago_is_open;
8     public int Pittsburg_is_open;
9     public int Nashville_is_open;
10 }
```

Note 1: Currently, the Bonsai platform only allows numerical fields (i.e., integers and doubles).

Note 2: The variables do not need to be assigned to any values. This will be done in the following steps.

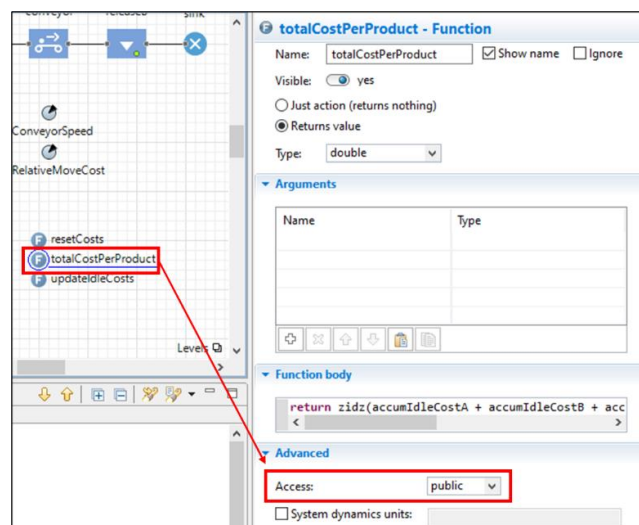
5. Modify the ModelConfig class by adding all the variables that you will want to initialize each episode with (i.e., each run of the simulation model).
 - a. This class may be left without variables declared if your initialization configuration happens as part of the simulation logic (e.g., on startup code of Main or from distributions that are assigned to the input parameters and variables).
 - b. Variables that are defined here are accessible in the “episodeStart” field of the Bonsai Connector object.
6. Modify the ModelObservation class by adding all the variables that are needed to define the observation space.
 - a. Variables that are defined here will be populated at each episode step with the code in the “getObservable” function.
7. Modify the ModelAction class by including all the variables that the brain will pass to your model
 - a. The “episodeStep” field of the Bonsai connector will use the values of these variables to modify your model in some defined way.

After configuring the Java classes in the Wrapper model, there are two items in the Wrapper agent type (seen in the image below) that need modification: the getObservable function and the Bonsai connector.

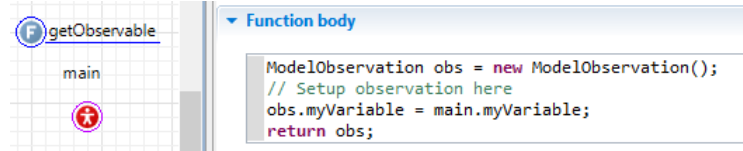


For the code that you write in `getObservable` function and Bonsai connector object, there are two important points:

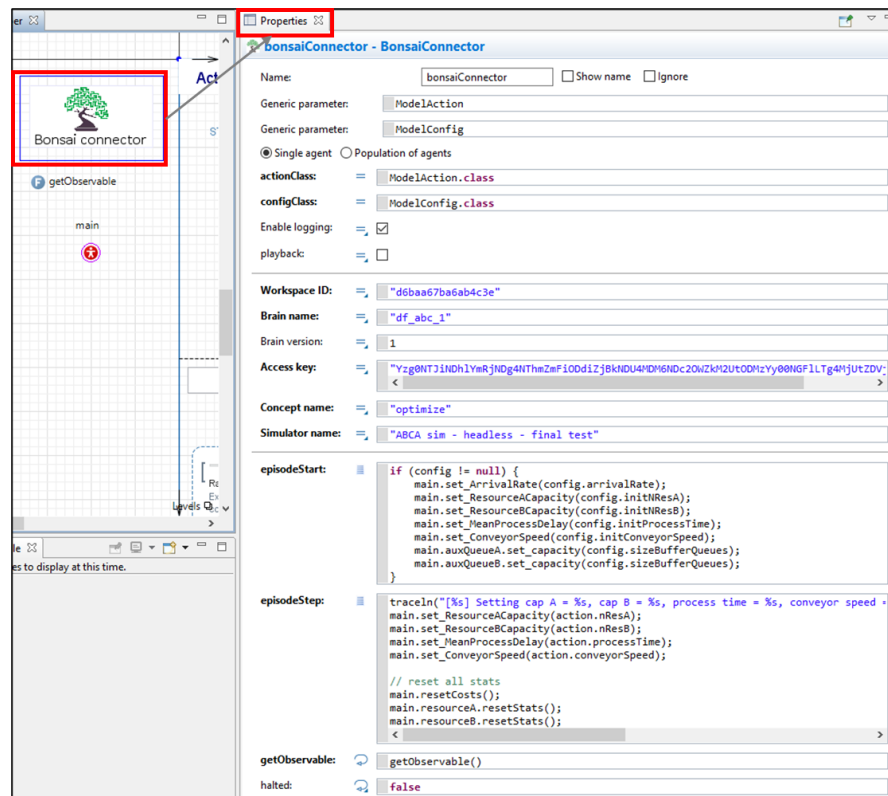
- i. Since the original model that you instantiated as an object in step 3 (e.g., `main`) is now accessible through the wrapper, any code that you write in `getObservable` function and the Bonsai connector need to reference the RL-ready model. In other words, any field or function in the original model are accessible through the root object (e.g., `main`) you added to the wrapper model. To access them, you have to prefix the fields or functions with the name of the dragged-in agent (in the above screenshot, “`main`” would be used).
 - a. For example, if I wanted to get access to a variable named “`myVariable`” in the root agent type that was named “`main`”, I would need to reference it using “`main.myVariable`”. This is applicable to the code that in the following steps.
- ii. Any function in the RL-ready model that needs to be called in the `getObservable` function or the Bonsai connector object should have their access type set to “`public`” (by default, it’s set to “`default`”).
 - a. To do so, first open the properties of the functions in your model, and in their Properties window, under the Advanced section, change the Access to “`public`” from the dropdown list. An example of this is shown in the screenshot below.



8. Considering the points mentioned above, modify the `getObservable` function to associate each element of the observation space (specified in `ModelObservation` class) with a proper output from the model. Using the example in point 'i' above, if a variable were declared in the `ModelObservation` class of the same name, the function body would consist of the code shown in the following image:



9. Click on the Bonsai connector object and fill the fields in its Properties (an example is shown in the picture below)



- The Workspace ID and Access key fields are available from the settings of your Bonsai account
- The brain name, brain version, and concept name fields are dependent on your desired configuration inside of the Bonsai webpage
- The Simulator name field is what will appear under 'Simulators' if training is done locally
- In the "episodeStart" field of the Bonsai Connector, you have access to a local variable called "config" which is an object built with the same values as defined in the `ModelConfig` and whose values will be provided by the Bonsai brain. This field will be triggered at the start of each episode before the model begins running. An example of such setup is shown in the picture below.

```

Simulator name: ABCA sim - headless - final test
Use: self; config

episodeStart:
    if (config != null) {
        main.set_ArrivalRate(config.arrivalRate);
        main.set_ResourceACapacity(config.initNResA);
        main.set_ResourceBCapacity(config.initNResB);
        main.set_MeanProcessDelay(config.initProcessTime);
        main.set_ConveyorSpeed(config.initConveyorSpeed);
        main.auxQueueA.set_capacity(config.sizeBufferQueues);
        main.auxQueueB.set_capacity(config.sizeBufferQueues);
    }

```

- e. In the “episodeStep” field, you have access to a local variable called “action” which consists of elements defined from your ModelAction class and whose values will be provided by the Bonsai brain (with its desired action to take). You should update the model in accordance with these fields. In addition, you can update other elements of your model not directly related to the action values (e.g., resetting some statistics gathering elements).

```

main.auxQueueB.set_capacity(config.sizeBufferQueues);
Use: self; action

episodeStep:
    println("[%s] Setting cap A = %s, cap B = %s, process time = %s, conv...);
    main.set_ResourceACapacity(action.nResA);
    main.set_ResourceBCapacity(action.nResB);
    main.set_MeanProcessDelay(action.processTime);
    main.set_ConveyorSpeed(action.conveyorSpeed);

    // reset all stats
    main.resetCosts();
    main.resourceA.resetStats();
    main.resourceB.resetStats();

```

- f. In the “halted” field is where you define the episode ending condition. This should be some time-based end point. Additionally, it can also be condition-based (e.g., the system reaches some capacity). If you want this to be defined in your Inking code, the halted condition should be set to the Java keyword false.
10. In case your RL-ready model uses the AnyLogic built-in database, you will need to migrate it to the Wrapper model. For this, you have two options (choose one):
- First, open both model directories, then close both models in AnyLogic and then copy the database folder from the original model folder and paste it in the wrapper model folder.
 - Inside the original model, go to the database object properties. Then, use the "back up database" option and save to the backup file on your hard drive. In the database object properties of the wrapper model, use the "restore database" option and select the file. In this method, you do not need to close the models. Afterwards, you can delete the file from the hard drive.

If you want to export the model and upload it to the Bonsai platform for scaling purposes, you just need to export the “HeadlessExperiment” of the wrapper model. AnyLogic will automatically bundle the two models into one package that includes everything that is required. In the exported model folder, you can safely delete the “chromium” folder to save some hard drive space. Zip the exported folder and upload it from within the Bonsai UI.