SCS V4.0.0

USER DOCUMENTATION

Contents

[2. What is SCS? 3](#_Toc68333980)

[2.1. Typical workflow using SCS 3](#_Toc68333981)

[2.2. SCS capabilities 3](#_Toc68333982)

[3. Requirements: 3](#_Toc68333983)

[4. How to install SCS? 3](#_Toc68333984)

[5. Main Concepts 5](#_Toc68333985)

[5.1. Summary 5](#_Toc68333986)

[5.2. SCS conceptual diagram 5](#_Toc68333987)

[5.3. SCS main loop (pseudocode) 6](#_Toc68333988)

[6. Running a model 7](#_Toc68333989)

[6.1. Command Line 7](#_Toc68333990)

[6.2. Eclipse IDE 8](#_Toc68333991)

[7. Experiment Files 9](#_Toc68333992)

[8. Configuration Files 9](#_Toc68333993)

[9. Simulator 10](#_Toc68333994)

[9.1. Abstract Simulator 10](#_Toc68333995)

[9.2. Abstract Simulator 10](#_Toc68333996)

[9.3. SCS\_Simulator 10](#_Toc68333997)

[9.4. Script priorities 10](#_Toc68333998)

[9.5. Script class diagrams 10](#_Toc68333999)

[10. Creating a new model 11](#_Toc68334000)

[11. Robot 11](#_Toc68334001)

[11.1. Commands 11](#_Toc68334002)

[11.2. Modules 11](#_Toc68334003)

[11.3. Proxies 11](#_Toc68334004)

[12. GUI support 11](#_Toc68334005)

[12.1. SCS’s GUI (swing) 11](#_Toc68334006)

[12.1.1. Panels 11](#_Toc68334007)

[12.2. Adding support for other GUIs 11](#_Toc68334008)

[13. Physics Simulator 11](#_Toc68334009)

[13.1. SCS simulator 11](#_Toc68334010)

[13.2. Adding support for external simulators 11](#_Toc68334011)

[14. Globals 12](#_Toc68334012)

[15. TODO 12](#_Toc68334013)

# What is SCS?

SCS is a lightweight software framework that allows to define and execute robot experiments (originally implemented for simulating rat experiments). The framework was developed with the following goals in mind:

* **Speed**: simulations need to run as fast as possible.
* **Ease of use**: the framework should be simple to learn and use
* **Extensibility**: the framework should be easily extended to be used with any simulators, robots (both real and simulated) and GUIs.
* **Code reutilization**: the code should be highly reusable allowing users to define their experiments with as few lines as possible.

## Typical workflow using SCS

1. Define hypothesis to be tested.
2. Develop parametric cognition model(s) based on the hypothesis.
3. Define parametric experiments to be performed.
4. Define configuration file (table specifying parameters for all executions of the experiment).
5. Run all configurations (simulations) and log results.
6. Post-process results: create plots and perform hypothesis testing.

## SCS capabilities

* Define parametrizable robot experiments using xml files.
* Use of configuration files for running parallel executions.
* Promotes reusability of robot modules, avoiding reimplementing same functionality for each robot.
* Seamlessly switch between real robots and different simulators.
* Choose the simulator that best suits your needs. Current supported simulators:
  + SCS\_simulator (SCS’s default bare metal kinematic simulator, runs very fast but provides minimal functionality).
  + Webots – Soon to be implemented.
* Use SCS’s basic GUI / plotting tools or easily create your own.

# Requirements:

SCS requires java 10 or later.

Optional scripts (recommended) require python with numpy and pandas. **NOTE**: DEPRECATED, scripts removed from SCS.

# How to install SCS?

SCS does not need to be installed. To use the framework, you only need to download and add the jar to your class path. If using maven (recommended), you only need to add the following lines of code to your pom file (replacing the version number by the desired [SCS release version](https://github.com/biorobaw/scs/tags)):

<repositories>

<repository>

<id>scs</id>

<url>http://raw.github.com/biorobaw/scs</url>

</repository>

</repositories>

<dependencies>

<dependency>

<groupId>com.github.biorobaw.scs</groupId>

<artifactId>scs</artifactId>

<version>0.1.0</version>

</dependency>

</dependencies>

# Main Concepts

## Summary

As mentioned in section 2, SCS is a framework meant for simplifying / streamlining the process of performing robot experiments and hypothesis testing with special emphasis in the context of spatial cognition. As such, the framework revolves around the concepts of “subject” (a.k.a. model), “robot”, “experiment” and “configuration”.

In SCS, a “subject” is an instantiation of an AI model used to control a robot typically based on some belief of how the brain works. The model implements high-level control logic and delegates lower-level routines to the robot itself.

Since SCS was developed with the idea of portability between real robots and different simulators, to reuse as much code as possible, a robot is defined as a set of modules and proxies. Modules allow writing routines that are independent of the simulator/robot in use, while proxies define the methods to communicate with the actual robots.

Next, an “experiment” represents a set of “trials” to be performed by a set of “subjects” defined in an “experiment file”. Here, a “trial” represents a set of identical “episodes” that the subjects must perform. For example, an experiment file could define 2 trials and 1 subject. The first trial could have a robot map a maze during episodes from different random positions, while the second trial could have the robot navigate towards different goals for episodes.

Finally, since typical SCS workflows require the experimenter to compare results of the experiment under different circumstances (i.e., perform hypothesis testing), experiment files are parametrized, and a complete instantiation of those parameters is referred to as a “configuration”. To streamline the process of hypothesis testing, and to simplify running parallel executions of the experiment in computer clusters, SCS uses “configuration files” which are basically CSV files defining all configurations to be executed.

## SCS conceptual diagram

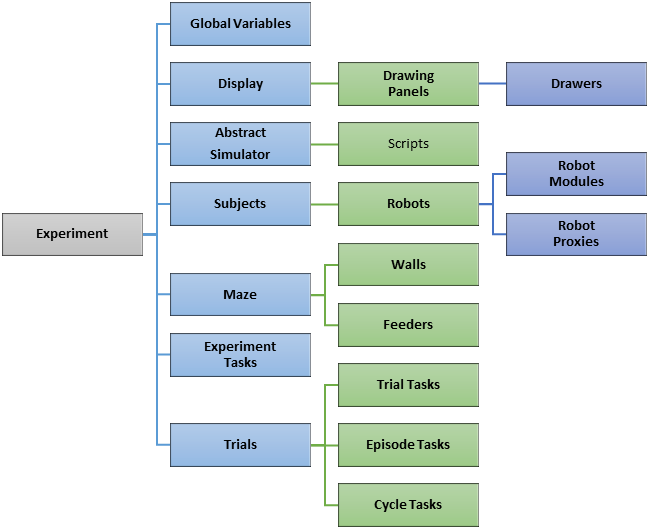


Figure – SCS main concepts

## SCS main loop (pseudocode)

1. Load simulation:
   1. Process parameters, set globals, and create log structure.
      1. Parse command arguments and store them as globals.
      2. Read params from config from file (as specified by command arguments), store them as globals.
      3. Read variables from experiment file, store them as globals. If variable already defined, do not overwrite it.
      4. Initialize other globals.
      5. Set random seed.
      6. Create log folders if specified in command argument.
   2. Load simulator script
   3. Load display script
   4. Load Maze
   5. Load trials for current group
   6. Load experiment tasks
   7. Load saved state // has to be reimplemented.
2. Run simulation.
   1. Signal new experiment
   2. For each trial
      1. Signal new trial
      2. For each episode
         1. Signal new episode
         2. Initialize simulation scheduler
            1. Clear scheduler
            2. Schedule cycle tasks
            3. Schedule subjects (robot high level controllers)
            4. Schedule robots (robot low level controllers)
            5. Schedule display
         3. Run simulation (scheduler)
            1. While not end episode:

Run scripts with scheduled time < current time + // script order depends on priority.

If simulation is paused, wait for “resume”

Advance time by (do physics simulation)

Signal events // partially implemented.

* + - 1. Signal end episode
    1. Signal end trial
  1. Signal experiment end

# Running a model

## Command Line

To run a model, execute the following command:

java -cp **CLASSPATH** com.github.biorobaw.scs.Main **ARGS**

ARGS = CONFIG\_FILE CONFIG\_NUM LOG\_FOLDER [CREATE\_FOLDERS]

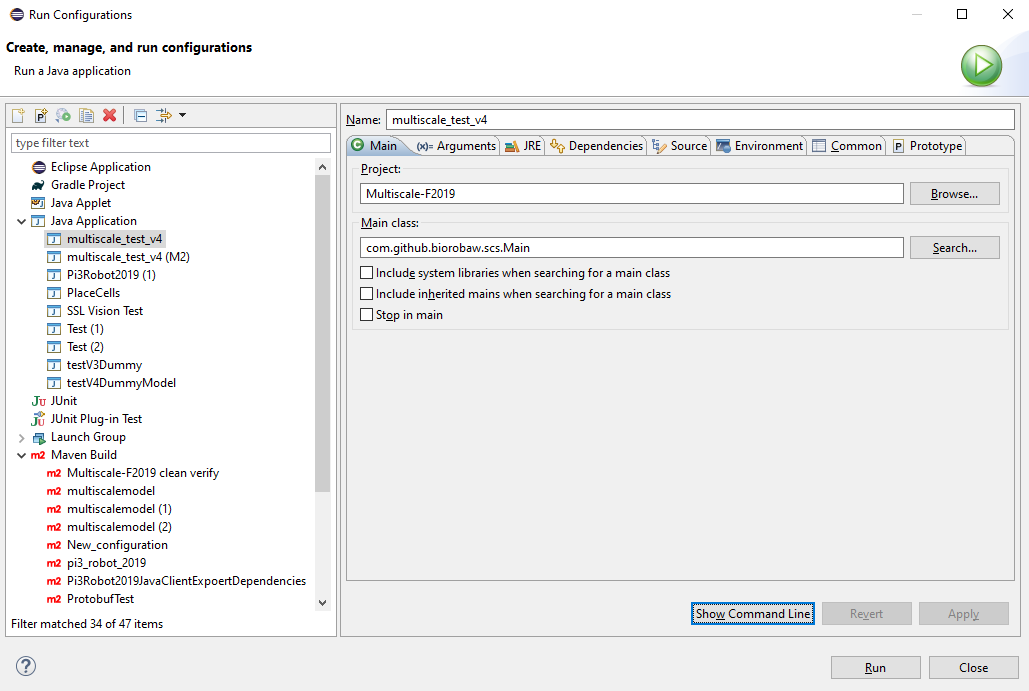
Where:

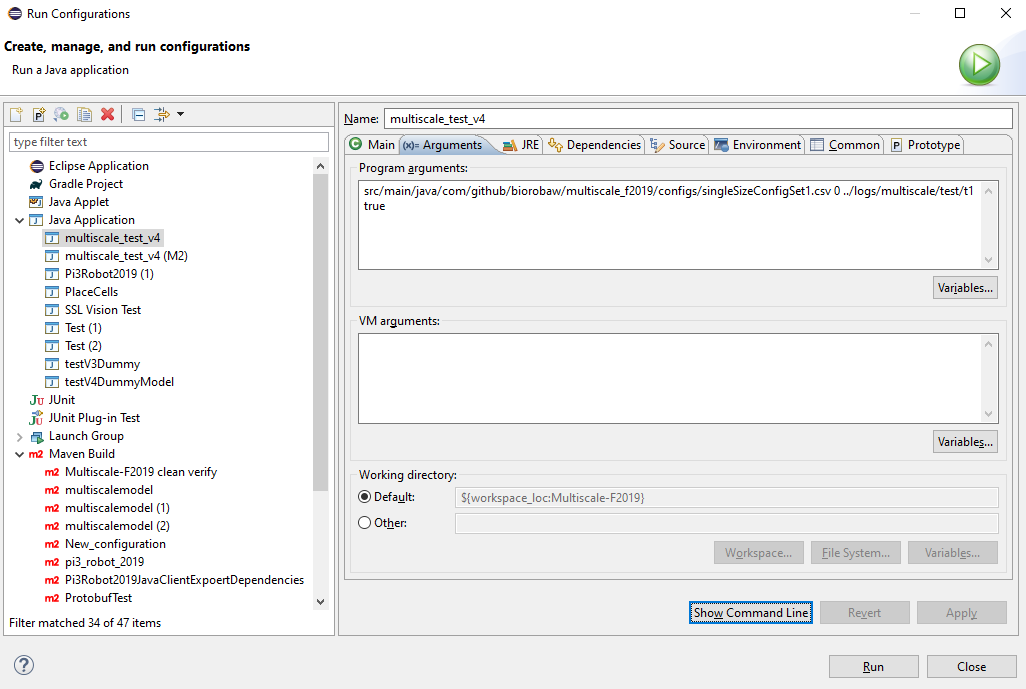
|  |  |
| --- | --- |
| **CLASSPATH** | Defines the java classpath[[1]](#footnote-1), scs should be included in the classpath. |
| **CONFIG\_FILE** | Path to a csv config file (see section “Configuration Files”). |
| **CONFIG\_NUM** | Row of the configuration file to be loaded (see section “Configuration Files”). The row number is zero based (i.e.: starts from 0). |
| **LOG\_FOLDER** | The path to store the simulation logs. |
| **CREATE\_FOLDERS** | Optional boolean argument specifying whether the log folders should be created before performing the experiment. Omitting the argument defaults to false. When running multiple concurrent simulations, you want to manually create the log folders (see “Running on USF’s cluster” for instructions on how to do so) and either omit the argument or set it to false. |

As a result of the command, an instance of SCS will start executing the experiment defined by the specified line in the specified csv configuration file.

## Eclipse IDE

To run a model from Eclipse, a Java Application run configuration must be specified. The following images show a sample run configuration defined in eclipse in a project using SCS. Note the main class and the arguments in the images correspond with the arguments described in section 6.1.





# Experiment Files

Describe experiment files in this section.

# Configuration Files

Configuration files are csv files that define a set of simulations to be performed. Although these files are meant for instantiating multiple parallel simulations in a cluster, their use was made compulsory to streamline the instantiation method.

A configuration file defines a table of parameters, where the first line defines the column headers and each line thereafter represents a row in the table. Note that columns must be separated by any type of whitespace. It is important not to use commas to separate columns since commas are used to parse list/array arguments. Also, blank rows are not currently supported, and all elements in the table are case sensitive.

To write a configuration file, you need to specify at least the following 4 columns:

* **experiment –** defines the path to an xml file describing the experiment to be performed (see section “Experiment Files” for more details).
* **config –** defines a name for a specific configuration of the experiment
* **group –** defines a group of subjects (from the experiment file) to be executed
* **run\_id –** defines an id to store the results of this simulation. Logging is done to LOG\_FOLDER/config/run\_id

Other columns can be defined as required. The order in which the columns are defined is irrelevant as long all rows are consistent with the headers. Python scripts that simplify the generation of configuration files are available in the “scripts” folder found in scs top folder (see section “scripts” for more information).

At the start of a simulation, a map is generated mapping each column header to the values specified by the line provided as a command argument (see “Running a model” to see the command arguments).

The following is a sample configuration file found used in the example model “dummy\_model” (src/main/java/github/biorobaw/scs/examples/dummy\_model/configs.csv):



# Simulator

SCS was created with extensibility in mind, thus the main SCS project provides only an AbstractSimulator and a basic implementation of the abstract simulator.

## Abstract Simulator

## Abstract Simulator

## SCS\_Simulator

SCS has a bare metal simulator, that, in its core, it is basically a script scheduler. In SCS almost everything is a script, including GUIs, robots, high and low level controllers, proxies, tasks performed by the experimenter, etc.

To run a simulation, a bunch of scripts are scheduled for execution before the start of each episode. After the initial scheduling, the scripts are executed according to the priority assigned to each script either until the scheduler runs out of scripts to execute, or until the “end episode” flag is raised.

## Script priorities

## Script class diagrams

# Creating a new model

Describe how to create a new model.

# Robot

## Commands

## Modules

## Proxies

# GUI support

## Display framework

### Display

### Panels

### Drawers

## Available Displays

### Display Swing

### Display JavaFX

## Adding support for other GUIs

# Physics Simulator

## SCS simulator

## Adding support for external simulators

# Globals

|  |  |
| --- | --- |
| “baseLogPath” | Path to experiment log folder |
| “logPath” | Path to the current configuration log folder |
| “configFile” | Path to configuration file specified in command arguments |
| “configLine” | Loaded row of the configuration file specified in command arguments |
| “create\_logs” | Boolean command argument, specifying whether to create logging folders. |
|  |  |
| “config” | Name of the config for logging purposes. |
| “experiment” | Path to experiment file defined in the specified row of the configuration file. |
| “group” | The group to be executed from the experiment file. |
| “run\_id” | The id for the current execution for the specified experimental group. |
|  |  |
| “maze” | Path of the loaded maze if specified in the experiment file. |
| “trial” | The id of the trial being executed. |
| “trial\_episodes” | The maximum number of episodes of the trial being executed. |
| “episode” | The number of the episode being executed. |
| “cycle” | The current simulation cycle being executed. |
| “seed” | SCS’s random number generator’s seed. |

# TODO

|  |  |
| --- | --- |
| Status | Task |
|  | Remove R scripts folder |
|  | Make “platform” the main branch |
|  |  |

sssss

1. See [java documentation](https://docs.oracle.com/javase/tutorial/essential/environment/paths.html) for more information about the class path. [↑](#footnote-ref-1)