
pysyncrosim Documentation

Release 1.0.16

ApexRMS

Jun 16, 2022

CONTENTS:

1	Installation	3
1.1	Dependencies	3
1.2	Using conda	3
1.3	Using pip	4
2	Quickstart	5
2.1	Overview of SyncroSim	5
2.2	Overview of <code>pysyncrosim</code>	5
2.3	SyncroSim Package: <i>helloworldTime</i>	6
2.4	Set Up	6
2.5	Create a Modeling Workflow	9
2.6	Run Scenarios	14
2.7	View Results	14
2.8	Working with Multiple Scenarios	15
2.9	Access Model Metadata	17
2.10	Backup your Library	18
2.11	<code>pysyncrosim</code> and the SyncroSim Windows User Interface	19
2.12	SyncroSim Package Development	20
3	Running <code>pysyncrosim</code> in Spyder	21
4	API Reference	23
4.1	<code>pysyncrosim.environment</code>	23
4.2	<code>pysyncrosim.helper</code>	23
4.3	<code>pysyncrosim.library.Library</code>	24
4.4	<code>pysyncrosim.project.Project</code>	25
4.5	<code>pysyncrosim.raster.Raster</code>	25
4.6	<code>pysyncrosim.scenario.Scenario</code>	26
4.7	<code>pysyncrosim.session.Session</code>	27
5	Support	29
6	Indices and tables	31
	Index	33

pysyncrosim is the Python interface to [SyncroSim](#)

INSTALLATION

`pysyncrosim` can be installed using either the `conda` or `pip` package managers. `conda` is a general package manager capable of installing packages from many sources, but `pip` is strictly a Python package manager. While the installation instructions below are based on a Windows 10 operating system, similar steps can be used to install `pysyncrosim` for Linux.

Before beginning the installation of `pysyncrosim`, make sure you have the latest release of [SyncroSim](#) installed.

1.1 Dependencies

`pysyncrosim` was tested and developed using **Python 3.8** and **SyncroSim 2.3.10**. Because `pysyncrosim` uses `rasterio` for integrating spatial data, it also requires a C library dependency: `GDAL >=2.3`.

- `python>=3.8`
- `pandas=1.3.2`
- `numpy=1.21.2`
- `rasterio>=1.2.6`

1.2 Using conda

Follow these steps to get started with `conda` and use `conda` to install `pysyncrosim`.

1. Install `conda` using the Miniconda or Anaconda installer (in this tutorial we use Miniconda). To install Miniconda, follow [this link](#) and under the **Latest Miniconda Installer Links**, download Miniconda for your operating system. Open the Miniconda installer and follow the default steps to install `conda`. For more information, see the [conda documentation](#).
2. To use `conda`, open the command prompt that was installed with the Miniconda installer. To find this prompt, type “anaconda prompt” in the **Windows Search Bar**. You should see an option appear called **Anaconda Prompt (miniconda3)**. Select this option to open a command line window. All code in the next steps will be typed in this window.
3. You can either install `pysyncrosim` and its dependencies into your base environment, or set up a new `conda` environment (recommended). Run the code below to set up and activate a new `conda` environment called “myenv” that uses Python 3.8.

```
# Create new conda environment
conda create -n myenv python=3.8

# Activate environment
conda activate myenv
```

You should now see that “(base)” has been replaced with “(myenv)” at the beginning of each prompt.

4. Set the package channel for conda. To be able to install `pysyncrosim`, you need to access the `conda-forge` package channel. To configure this channel, run the following code in the Anaconda Prompt.

```
# Set conda-forge package channel
conda config --add channels conda-forge
```

5. Install `pysyncrosim` using `conda install`. Installing `pysyncrosim` will also install its dependencies: `pandas`, `numpy`, and `rasterio`.

```
# Install pysyncrosim
conda install pysyncrosim
```

`pysyncrosim` should now be installed and ready to use!

1.3 Using pip

Use `pip` to install `pysyncrosim` to your default python installation. You can install Python from www.python.org. You can also find information on how to install `pip` from the [pip documentation](#).

Install `pysyncrosim` using `pip install`. Installing `pysyncrosim` will also install its dependencies: `pandas`, `numpy`, and `rasterio`.

```
# Make sure you are using the latest version of pip
pip install --upgrade pip

# Install pysyncrosim
pip install pysyncrosim
```


QUICKSTART

`pysyncrosim` is the Python interface to the [SyncroSim software framework](#), a program that structures and models your data. This tutorial will cover the basics of using the `pysyncrosim` package within the SyncroSim software framework.

To complete this tutorial, you must [install SyncroSim](#) and [install pysyncrosim](#). You will also need to install the [hel-loworldTime SyncroSim Package](#).

2.1 Overview of SyncroSim

[SyncroSim](#) is a software platform that helps you turn your data into forecasts. At the core of SyncroSim is an engine that automatically structures your existing data, regardless of its original format. SyncroSim transforms this structured data into forecasts by running it through a Pipeline of calculations (i.e. a suite of models). Finally, SyncroSim provides a rich interface to interact with your data and models, allowing you to explore and track the consequences of alternative “what-if” forecasting Scenarios. Within this software framework is the ability to use and create SyncroSim packages.

For more details consult the [SyncroSim online documentation](#).

2.2 Overview of `pysyncrosim`

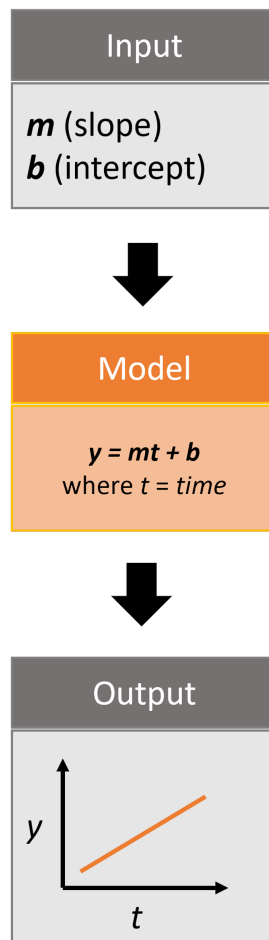
`pysyncrosim` is a Python package designed to facilitate the development of modeling workflows for the SyncroSim software framework. Using the `pysyncrosim` interface, simulation models can be added and run through SyncroSim to transform Scenario-based datasets into model forecasts. This Python package takes advantage of general features of SyncroSim, such as defining Scenarios with spatial or non-spatial inputs, running Monte Carlo simulations, and summarizing model outputs.

`pysyncrosim` requires SyncroSim 2.3.10 or higher.

2.3 SyncroSim Package: *helloworldTime*

helloworldTime was designed to be a simple package to show off some key functionalities of SyncroSim, including the ability to add timesteps to SyncroSim modeling workflows.

The package takes from the user two inputs, m and b , representing a slope and an intercept value. It then runs these input values through a linear model, $y=mt+b$, where t is *time*, and returns the y value as output.



2.4 Set Up

2.4.1 Install SyncroSim

Before using `pysyncrosim`, you will first need to [download and install](#) the SyncroSim software. Versions of SyncroSim exist for both Windows and Linux.

2.4.2 Installing and Loading Python Packages

Install `pysyncrosim` using either `conda install` or `pip install`. See the [Installation](#) page for more detailed installation instructions.

Then, in a new Python script, import `pysyncrosim`.

```
>>> import pysyncrosim as ps
```

2.4.3 Connecting Python to SyncroSim

The next step in setting up the Python environment for the `pysyncrosim` workflow is to create a `SyncroSim Session` instance in Python that provides the connection to your installed copy of the `SyncroSim` software. A new `Session` is created using the `Session()` class. The `Session` can be initialized with a path to the folder on your computer where `SyncroSim` has been installed. If no arguments are specified when the `Session` class is initialized, then the default install folder is used (Windows only).

```
# Load Session
>>> mySession = ps.Session()

# Load Session using path to SyncroSim Installation
>>> mySession = ps.Session(location = "path/to/install_folder")
```

You can check to see which version of `SyncroSim` your Python script is connected to by running the `version()` method.

```
# Check SyncroSim version
>>> mySession.version()
'Version is: 2.3.10'
```

2.4.4 Installing SyncroSim Packages

Finally, check if the `helloworldTime` package is already installed. Use the `packages()` method to first get a list of all currently installed `SyncroSim` Packages.

```
# Check which SyncroSim Packages are installed
>>> mySession.packages()
Empty DataFrame
Columns: [index, Name, Description, Version, Extends]
Index: []
```

Currently we do not have any packages installed! To see which packages are available from the `SyncroSim` package server, you can use the `installed = False` argument in the `packages()` method.

```
# Check out available SyncroSim Packages
>>> available_packages = mySession.packages(installed=False)
>>> available_packages.head()
   demosales \
0          dgsim
1    helloworld
2  helloworldPipeline
```

(continues on next page)

(continued from previous page)

```

3  helloworldSpatial
4  helloworldTime

Example SyncroSim Base Package of a very simple sales forecasting model \
0  Simulates demographics of wildlife populations
1  Example demonstrating how to create a package
2  Example using pipelines
3  Example using spatial data
4  Example using timesteps

1.1.0
0 2.3.0
1 1.0.1
2 1.0.0
3 1.0.2
4 1.0.0

```

Install `helloworldTime` using the `add_packages()` method from the `Session` class. This method takes a `Package` name as input and then queries the SyncroSim Package server for the specified `Package`.

```

# Install helloworldTime Package
>>> mySession.add_packages("helloworldTime")
['helloworldTime'] installed successfully

```

To install a package from a `.ssimpkg` file on your local computer rather than installing directly from the server, you can use the `Session` `add_packages()` method with the `packages` argument set to the filepath to the local `Package`.

```

# Install helloworldTime Package locally
>>> mySession.add_packages("path/to/helloworldTime.ssimpkg")

```

Now `helloworldTime` should be included in the `Package` list.

```

>>> mySession.packages()

```

	index	Name	Description	Version
↳ Extends				
	0	helloworldTime	Example demonstrating how to use timesteps	1.0.0
↳	NaN			

You can also update or remove a SyncroSim `Package` from your `Session` using the `update_packages()` method and the `remove_packages()` method.

```

# Update installed packages
>>> mySession.update_packages("myPackage")

# Remove installed packages
>>> mySession.remove_packages("myPackage")

```

2.5 Create a Modeling Workflow

When creating a new modeling workflow from scratch, we need to create class instances of the following scopes:

- Library
- Project
- Scenario

These classes are hierarchical, such that a Library can contain many Projects, and each Project can contain many Scenarios. All parameters or configurations set in a Library are inherited by all Projects within the Library, and all parameters or configurations set in a Project are inherited by all Scenarios within that Project.

2.5.1 Create a New Library

A SyncroSim [Library](#) is a file (with `.ssim` extension) that stores all of your model inputs and outputs. The format of each SyncroSim Library is unique to the SyncroSim Package with which it is associated. We create a new Library class instance using `library()` that is connected (through your Session) to a SyncroSim Library file.

```
# Create a new Library
>>> myLibrary = ps.library(name = "helloworldLibrary",
>>>                        session = mySession,
>>>                        package = "helloworldTime")

# Check Library information
>>> myLibrary.info
```

	Property	Value
0	Name:	helloworldLibrary
1	Owner:	NaN
2	Last Modified:	2021-09-10 at 3:13 PM
3	Size:	196 KB (200,704 B)
4	Read Only:	No
5	Package Name:	helloworldTime
6	Package Description:	Example demonstrating how to use timesteps
7	Current Package Version:	1.0.0
8	Minimum Package Version:	1.0.0
9	External input files:	helloworldLibrary.ssim.input
10	External output files:	helloworldLibrary.ssim.output
11	Temporary files:	helloworldLibrary.ssim.temp
12	Backup files:	helloworldLibrary.ssim.backup

We can also use the `library()` function to open an existing Library. For instance, now that we have created a Library called “helloworldLibrary.ssim”, we would simply specify that we want to open this Library using the `name` argument.

```
# Open existing Library
>>> myLibrary = ps.library(name = "helloworldLibrary")
```

Note that if you want to create a new Library file with an existing Library name rather than opening the existing Library, you can use `overwrite=True` when initializing the Library class instance.

2.5.2 Create a New Project

Each SyncroSim Library contains one or more SyncroSim **Projects**, each represented by an instance of class `Project` in `pysyncrosim`. Projects typically store model inputs that are common to all your Scenarios. In most situations you will need only a single Project for your Library; by default each new Library starts with a single Project named “Definitions” (with a unique `project_id = 1`). The `projects()` method of the `Library` class is used to both create and retrieve Projects for a specific Library.

```
# Create (or open) a Project in this Library
>>> myProject = myLibrary.projects(name = "Definitions")

# Check Project information
>>> myProject.info
    Property          Value
0   ProjectID         1
1     Name      Definitions
2     Owner           NaN
3 DateLastModified 2021-12-21 at 10:48 PM
4   IsReadOnly         No
```

2.5.3 Create a New Scenario

Finally, each SyncroSim Project contains one or more **Scenarios**, each represented by an instance of class `Scenario` in `pysyncrosim`.

Scenarios store the specific inputs and outputs associated with each Transformer in SyncroSim. SyncroSim models can be broken down into one or more of these Transformers. Each Transformer essentially runs a series of calculations on the input data to transform it into the output data. Scenarios can contain multiple Transformers connected by a series of Pipelines, such that the output of one Transformer becomes the input of the next.

Each Scenario can be identified by its unique `scenario_id`. The `scenarios()` method of class `Library` or class `Project` is used to both create and retrieve Scenarios. Note that if using the `Library` class to generate a new Scenario, you must specify the Project to which the new Scenario belongs if there is more than one Project in the Library.

```
# Create a new Scenario using the Library class instance
>>> myScenario = myLibrary.scenarios(name = "My First Scenario")

# Open the newly-created Scenario using the Project class instance
>>> myScenario = myProject.scenarios(name = "My First Scenario")

# Check Scenario information
>>> myScenario.info
    Property          Value
0   ScenarioID         1
1   ProjectID         1
2     Name      My First Scenario
3   IsResult           No
4   ParentID           NaN
5     Owner           NaN
6 DateLastModified 2021-09-10 at 3:13 PM
```

(continues on next page)

(continued from previous page)

7	IsReadOnly	No
8	MergeDependencies	No
9	IgnoreDependencies	NaN
10	AutoGenTags	NaN

2.5.4 View Model Inputs

Each SyncroSim Library contains multiple SyncroSim [Datasheets](#). A SyncroSim Datasheet is simply a table of data stored in the Library, and they represent the input and output data for Transformers. Datasheets each have a *scope*: either [Library](#), [Project](#), or [Scenario](#). Datasheets with a Library scope represent data that is specified only once for the entire Library, such as the location of the backup folder. Datasheets with a Project scope represent data that are shared over all Scenarios within a Project. Datasheets with a Scenario scope represent data that must be specified for each generated Scenario. We can view Datasheets of varying scopes using the `datasheets()` method from the Library, Project, and Scenario classes.

```
# View a summary of all Datasheets associated with the Scenario
>>> myScenario.datasheets()
   Package                                Name      Display Name
0 helloworldTime  helloworldTime_InputDatasheet  InputDatasheet
1 helloworldTime  helloworldTime_OutputDatasheet  OutputDatasheet
2 helloworldTime      helloworldTime_RunControl      Run Control
```

If we want to see more information about each Datasheet, such as the scope of the Datasheet or if it only accepts a single row of data, we can set the optional argument to `True`.

```
# View detailed summary of all Datasheets associated with a Scenario
>>> myScenario.datasheets(optional=True)
   Scope      Package                                Name      Display Name \
0 Scenario  helloworldTime  helloworldTime_InputDatasheet  InputDatasheet
1 Scenario  helloworldTime  helloworldTime_OutputDatasheet  OutputDatasheet
2 Scenario  helloworldTime      helloworldTime_RunControl      Run Control

   Is Single  Is Output
0         Yes         No
1         No         No
2         Yes         No
```

From this output we can see the the `RunControl` Datasheet and `InputDatasheet` only accept a single row of data (i.e. `Is Single = Yes`). This is something to consider when we configure our model inputs.

To view a specific Datasheet rather than just a DataFrame of available Datasheets, set the `name` parameter in the `datasheets()` method to the name of the Datasheet you want to view. The general syntax of the name is: “<name of package>_<name of Datasheet>”. From the list of Datasheets above, we can see that there are three Datasheets specific to the `helloworldTime` package.

```
# View the input Datasheet for the Scenario
>>> myScenario.datasheets(name = "helloworldTime_InputDatasheet")
Empty DataFrame
Columns: [m, b]
Index: []
```

Here, we are viewing the contents of a SyncroSim Datasheet as a Python pandas DataFrame. Although both SyncroSim Datasheets and pandas DataFrames are both represented as tables of data with predefined columns and an unlimited number of rows, the underlying structure of these tables differ.

2.5.5 Configure Model Inputs

Currently our input Scenario Datasheets are empty! We need to add some values to our input Datasheet (`InputDatasheet`) and run control Datasheet (`RunControl`) so we can run our model.

First, assign the contents of the input Datasheet to a new pandas DataFrame using the `Scenario datasheets()` method, then check the columns that need input values.

```
# Load input Datasheet to a new pandas DataFrame
>>> myInputDataframe = myScenario.datasheets(
>>>     name = "helloworldTime_InputDatasheet")

# Check the columns of the input DataFrame
>>> myInputDataframe.info()
<class 'pandas.core.frame.DataFrame'>
Index: 0 entries
Data columns (total 2 columns):
#   Column  Non-Null Count  Dtype
---  -
0    m         0 non-null    object
1    b         0 non-null    object
dtypes: object(2)
memory usage: 0.0+ bytes
```

The input Datasheet requires two values:

- m : the slope of the linear equation.
- b : the intercept of the linear equation.

Now we will update the input DataFrame. This can be done in many ways, such as creating a new pandas DataFrame with matching column names, or appending values as a dictionary to `myInputDataframe`.

For this example, we will append values to `myInputDataframe` using a Python dictionary and the pandas `append()` function. Note that in the previous section we discovered that the input Datasheets only accept a single row of values, so we can only have one value each for our slope (m) and intercept (b).

```
# Create input data dictionary
>>> myInputDict = {"m": 3, "b": 10}

# Append input data dictionary to myInputDataframe
>>> myInputDataframe = myInputDataframe.append(myInputDict,
>>>                                             ignore_index=True)

# Check values
>>> myInputDataframe
   m  b
0  3 10
```


2.5.6 Saving Modifications to Datasheets

Now that we have a complete DataFrame of input values, we will save this DataFrame to a SyncroSim Datasheet using the Scenario `save_datasheet()` method. The `save_datasheet()` method exists for the Library, Project, and Scenario classes, so the class method chosen depends on the scope of the Datasheet.

```
>>> myScenario.save_datasheet(name = "helloworldTime_InputDatasheet",
>>>                             data = myInputDataframe)
```

2.5.7 Configuring the RunControl Datasheet

There is one other Datasheet that we need to configure for our model to run. The RunControl Datasheet provides information about how many time steps to use in the model. Here, we set the minimum and maximum time steps for our model. Similar to above, we'll add this information to a Python dictionary and then add it to the RunControl Datasheet using the `pandas.append()` function. We need to specify data for the following two columns:

- *MinimumTimestep* : the starting time point of the simulation.
- *MaximumTimestep* : the end time point of the simulation.

```
# Load RunControl Datasheet to a ``pandas`` DataFrame
>>> runSettings = myScenario.datasheets(
>>>     name = "helloworldTime_RunControl")

# Check the columns of the RunControl DataFrame
>>> runSettings.info()
<class 'pandas.core.frame.DataFrame'>
Index: 0 entries
Data columns (total 2 columns):
#   Column                Non-Null Count  Dtype
---  -
0   MinimumTimestep      0 non-null      object
1   MaximumTimestep      0 non-null      object
dtypes: object(2)
memory usage: 0.0+ bytes

# Create RunControl data dictionary
>>> runControlDict = {"MinimumTimestep": 1, "MaximumTimestep": 10}

# Append RunControl data dictionary to RunControl DataFrame
>>> runSettings = runSettings.append(runControlDict, ignore_index=True)

# Check values
>>> runSettings
   MinimumTimestep  MaximumTimestep
0                 1                 10

# Save RunControl pandas DataFrame to a SyncroSim Datasheet
>>> myScenario.save_datasheet(name = "helloworldTime_RunControl",
>>>                             data = runSettings)
```

2.6 Run Scenarios

2.6.1 Setting Run Parameters

We will now run our Scenario using the Scenario `run()` method.

```
# Run the Scenario
>>> myResultsScenario = myScenario.run()
```

2.6.2 Checking the Run Log

For more information use the Scenario `run_log()` method. Note that this method can only be called when a Scenario is a *Results Scenario*.

```
# Get run details for My First Scenario
>>> myResultsScenario.run_log()

                                Run Log
0      STARTING SIMULATION: 2022-01-13 : 8:34:46 AM
1      Parent Scenario is: [1] My First Scenario
2      Result scenario is: [2] My First Scenario ([1]...
3      CONFIGURING: Primary
4      RUNNING: Primary
5      SIMULATION COMPLETE: 2022-01-13 : 8:34:54 AM
6      Total simulation time: 00:00:08
```

2.7 View Results

2.7.1 Results Scenarios

A Results Scenario is generated when a Scenario is run, and is an exact copy of the original Scenario (i.e. it contains the original Scenario's values for all input Datasheets). The Results Scenario is passed to the Transformer in order to generate model output, with the results of the Transformer's calculations then being added to the Results Scenario as output Datasheets. In this way the Results Scenario contains both the output of the run and a snapshot record of all the model inputs.

Check out the current Scenarios in your Library using the Library `scenarios()` method.

```
# Check Scenarios that currently exist in your Library
>>> myLibrary.scenarios()
  ScenarioID  ProjectID                Name \
0            1            1                My First Scenario
1            2            1 My First Scenario ([1] @ 13-Jan-2022 8:34 AM)

  IsResult
0         No
1         Yes
```

The first Scenario is our original Scenario, and the second is the Results Scenario with a time and date stamp of when it was run. We can also see some other information about these Scenarios, such as whether or not the Scenario is a result or not (i.e. `isResult` column).

2.7.2 Viewing Results

The next step is to view the output Datasheets added to the Result Scenario when it was run. We can load the result tables using the Scenario `datasheets()` method, and setting the name parameter to the Datasheet with new data added.

```
# Results of Scenario
>>> myOutputDataframe = myResultsScenario.datasheets(
>>>     name = "helloworldTime_OutputDatasheet")

# View results table
>>> myOutputDataframe.head()
  Iteration  Timestep    y
0         NaN         1  13.0
1         NaN         2  16.0
2         NaN         3  19.0
3         NaN         4  22.0
4         NaN         5  25.0
```

2.8 Working with Multiple Scenarios

You may want to test multiple alternative Scenarios that have slightly different inputs. To save time, you can copy a Scenario that you've already made, give it a different name, and modify the inputs. To copy a completed Scenario, use the Scenario `copy()` method.

```
# Check which Scenarios you currently have in your Library
>>> myLibrary.scenarios().Name
0
1    My First Scenario ([1] @ 13-Jan-2022 8:34 AM)
Name: Name, dtype: object

# Create a new Scenario as a copy of an existing Scenario
>>> myNewScenario = myScenario.copy("My Second Scenario")

# Make sure this new Scenario has been added to the Library
>>> myLibrary.scenarios().Name
0
1    My First Scenario ([1] @ 13-Jan-2022 8:34 AM)
2
Name: Name, dtype: object
```

To edit the new Scenario, let's first load the contents of the input Datasheet and assign it to a new pandas DataFrame using the Scenario `datasheets()` method. We will set the `empty` argument to `True` so that instead of getting the values from the existing Scenario, we can start with an empty DataFrame again.

```
# Load empty input Datasheets as a Pandas DataFrame
>>> myNewInputDataframe = myNewScenario.datasheets(
>>>     name = "helloworldTime_InputDatasheet", empty = True)

# Check that we have an empty DataFrame
>>> myNewInputDataframe.info()
<class 'pandas.core.frame.DataFrame'>
```

(continues on next page)

(continued from previous page)

```

Index: 0 entries
Data columns (total 2 columns):
#   Column  Non-Null Count  Dtype
---  -
0   m        0 non-null      object
1   b        0 non-null      object
dtypes: object(2)
memory usage: 0.0+ bytes

```

Now, all we need to do is add some new values the same way we did before, using the pandas `append()` function.

```

# Create an input data dictionary
>>> newInputDict = {"m": 4, "b": 10}

# Append new input data dictionary to new input DataFrame
>>> myNewInputDataframe = myNewInputDataframe.append(newInputDict,
>>>                                                    ignore_index=True)

# View the new inputs
>>> myNewInputDataframe
   m  b
0  4 10

```

Finally, we will save the updated DataFrame to a SyncroSim Datasheet using the Scenario `save_datasheet()` method.

```

# Save pandas DataFrame to a SyncroSim Datasheet
>>> myNewScenario.save_datasheet(name = "helloworldTime_InputDatasheet",
>>>                               data = myNewInputDataframe)

```

We will keep the RunControl Datasheet the same as the first Scenario.

2.8.1 Run Scenarios

We now have two SyncroSim Scenarios. We can run all the Scenarios using Python list comprehension.

```

# Create a List of Scenarios
>>> myScenarioList = [myScenario, myNewScenario]

# Run all Scenarios
>>> myResultsScenarioAll = [scn.run() for scn in myScenarioList]

```

2.8.2 View Results

From running many Scenario at once we get a list of Result Scenarios. To view the results, we can use the Scenario `datasheets()` method on the indexed list.

```

# View results of second Scenario
>>> myResultsScenarioAll[1].datasheets(
>>>     name = "helloworldTime_OutputDatasheet")
  Iteration  Timestep    y

```

(continues on next page)

(continued from previous page)

0	NaN	1	14.0
1	NaN	2	18.0
2	NaN	3	22.0
3	NaN	4	26.0
4	NaN	5	30.0
5	NaN	6	34.0
6	NaN	7	38.0
7	NaN	8	42.0
8	NaN	9	46.0
9	NaN	10	50.0

2.8.3 Identifying the Parent Scenario of a Results Scenario

If you have many alternative Scenarios and many Results Scenarios, you can always find the parent Scenario that was run in order to generate the Results Scenario using the Scenario `parent_id` attribute.

```
# Find parent ID of first Results Scenario
>>> myResultsScenarioAll[0].parent_id
1.0

# Find parent ID of second Results Scenario
>>> myResultsScenarioAll[1].parent_id
3.0
```

2.9 Access Model Metadata

2.9.1 Getting SyncroSim Class Information

Retrieve information about your Library, Project, or Scenario using the `info` attribute.

```
# Retrieve Library information
>>> myLibrary.info

      Property                                     Value
0      Name:                                     helloworldLibrary
1      Owner:                                     NaN
2      Last Modified:                             2021-09-10 at 3:13 PM
3      Size:                                       196 KB (200,704 B)
4      Read Only:                                  No
5      Package Name:                               helloworldTime
6      Package Description: Example demonstrating how to use timesteps
7      Current Package Version:                    1.0.0
8      Minimum Package Version:                    1.0.0
9      External input files:                       helloworldLibrary.ssim.input
10     External output files:                      helloworldLibrary.ssim.output
11     Temporary files:                            helloworldLibrary.ssim.temp
12     Backup files:                               helloworldLibrary.ssim.backup

# Retrieve Project information
```

(continues on next page)

(continued from previous page)

```

>>> myProject.info
      Property          Value
0      ProjectID          1
1          Name      Definitions
2          Owner          NaN
3 DateLastModified 2021-12-21 at 10:48 PM
4      IsReadOnly          No

# Retrieve Scenario information
>>> myScenario.info
      Property          Value
0      ScenarioID          1
1      ProjectID          1
2          Name      My First Scenario
3      IsResult          No
4      ParentID          NaN
5          Owner          NaN
6 DateLastModified 2021-09-10 at 3:13 PM
7      IsReadOnly          No
8 MergeDependencies          No
9 IgnoreDependencies          NaN
10     AutoGenTags          NaN

```

The following attributes can also be used to get useful information about a Library, Project, or Scenario instance:

- `name`: used to retrieve or assign a name.
- `owner`: used to retrieve or assign an owner.
- `date_modified`: used to retrieve the timestamp when the last changes were made.
- `readonly`: used to retrieve or assign the read-only status.
- `description`: used to retrieve or add a description.

You can also find identification numbers of Projects or Scenarios using the following attributes:

- `project_id`: used to retrieve the Project Identification number.
- `scenario_id`: used to retrieve the Scenario Identification number.

2.10 Backup your Library

Once you have finished running your models, you may want to backup the inputs and results into a zipped `.backup` subfolder. First, we want to modify the Library Backup Datasheet to allow the backup of model outputs. Since this Datasheet is part of the built-in SyncroSim core, the name of the Datasheet has the prefix “core”. We can get a list of all the core Datasheets with a Library scope using the `Library.datasheets()` method with `summary` set to “CORE”.

```

# Find all Library-scoped Datasheets
>>> myLibrary.datasheets(summary = "CORE")
      Package          Name          Display Name
0      core      core_Backup          Backup
1      core      core_Config          Conda Configuration
2      core      core_LNGPackage  Last Known Good Packages

```

(continues on next page)

(continued from previous page)

```

3      core      core_Multiprocessing      Multiprocessing
4      core      core_Options              Options
5      core      core_ProcessorGroupOption  Processor Group Options
6      core      core_ProcessorGroupValue  Processor Group Values
7      core      core_PyConfig             Python Configuration
8      core      core_RConfig              R Configuration
9      core      core_Settings              Settings
10     core      core_SysFolder             Folders
11     coretime   coretime_Options                       Spatial Options

# Get the current values for the Library's Backup Datasheet
>>> myDataframe = myLibrary.datasheets(name = "core_Backup")

# View current values for the Library's Backup Datasheet
>>> myDataframe
   IncludeInput  IncludeOutput  BeforeUpdate
0             Yes             NaN             Yes

# Add IncludeOutput to the Library's Backup Datasheet
>>> myDataframe["IncludeOutput"] = "Yes"

# Save the pandas DataFrame to a SyncroSim Datasheet
>>> myLibrary.save_datasheet(name = "core_Backup", data = myDataframe)

# Check to make sure IncludeOutput is now set to "Yes"
>>> myLibrary.datasheets(name = "core_Backup")

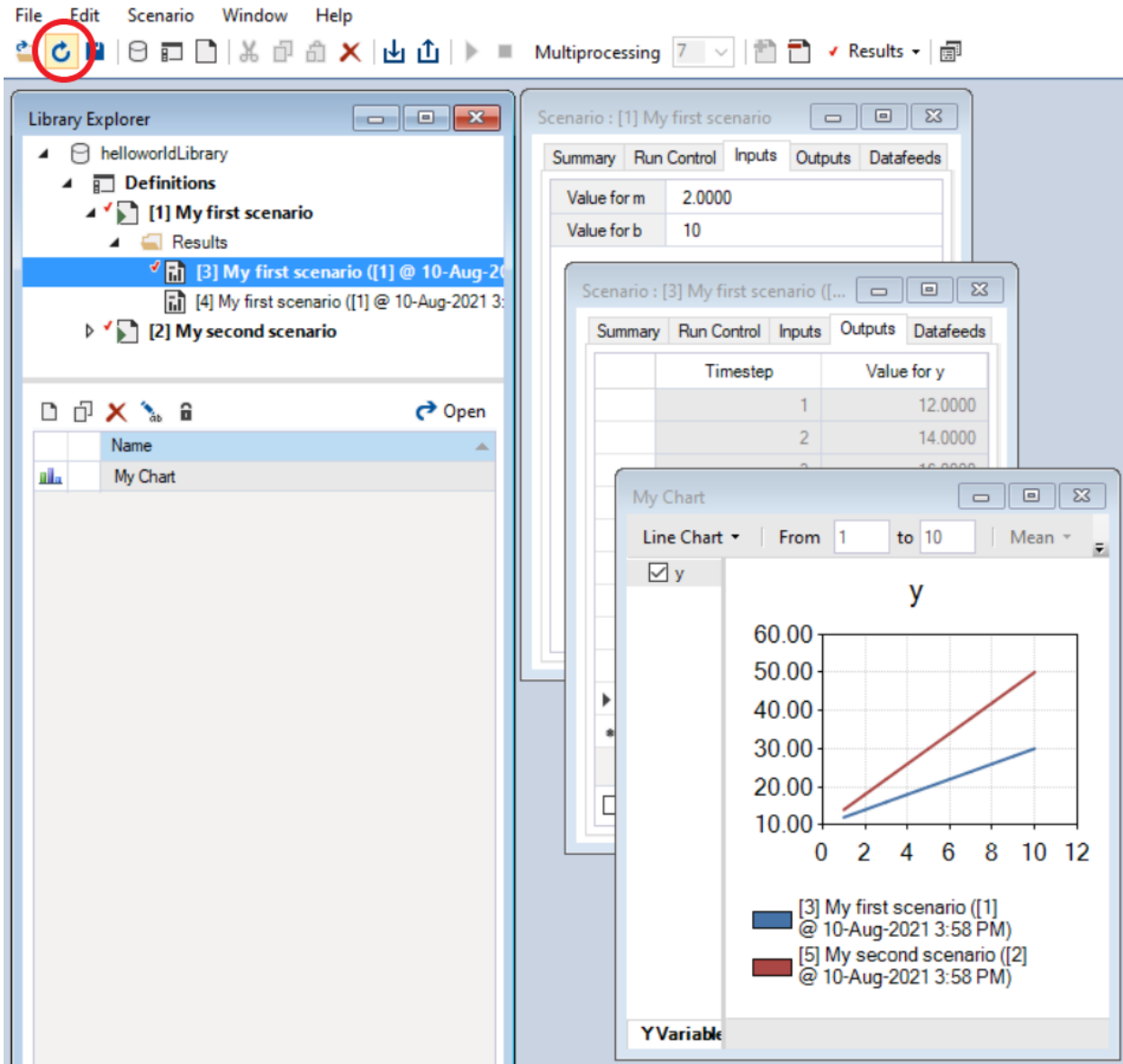
```

Now, you can use the Library backup() method to backup your Library.

```
>>> myLibrary.backup()
```

2.11 pysyncrosim and the SyncroSim Windows User Interface

It can be useful to work in both pysyncrosim and the SyncroSim Windows User Interface at the same time. You can easily modify Datasheets and run Scenarios in pysyncrosim, while simultaneously refreshing the Library and plotting outputs in the User Interface as you go. To sync the Library in the User Interface with the latest changes from the pysyncrosim code, click the refresh icon (circled in red below) in the upper tool bar of the User Interface.



2.12 SyncroSim Package Development

If you wish to design SyncroSim packages using python and pysyncrosim, you can follow the [Creating a Package](#) and [Enhancing a Package](#) tutorials on the SyncroSim documentation website.

Note: SyncroSim v2.3.10 is required to develop python-based SyncroSim packages.

RUNNING PYSYNCRSIM IN SPYDER

If using conda, the spyder IDE is easy to install and straightforward to use.

1. First, install spyder either in your base environment or in your conda environment using the following code.

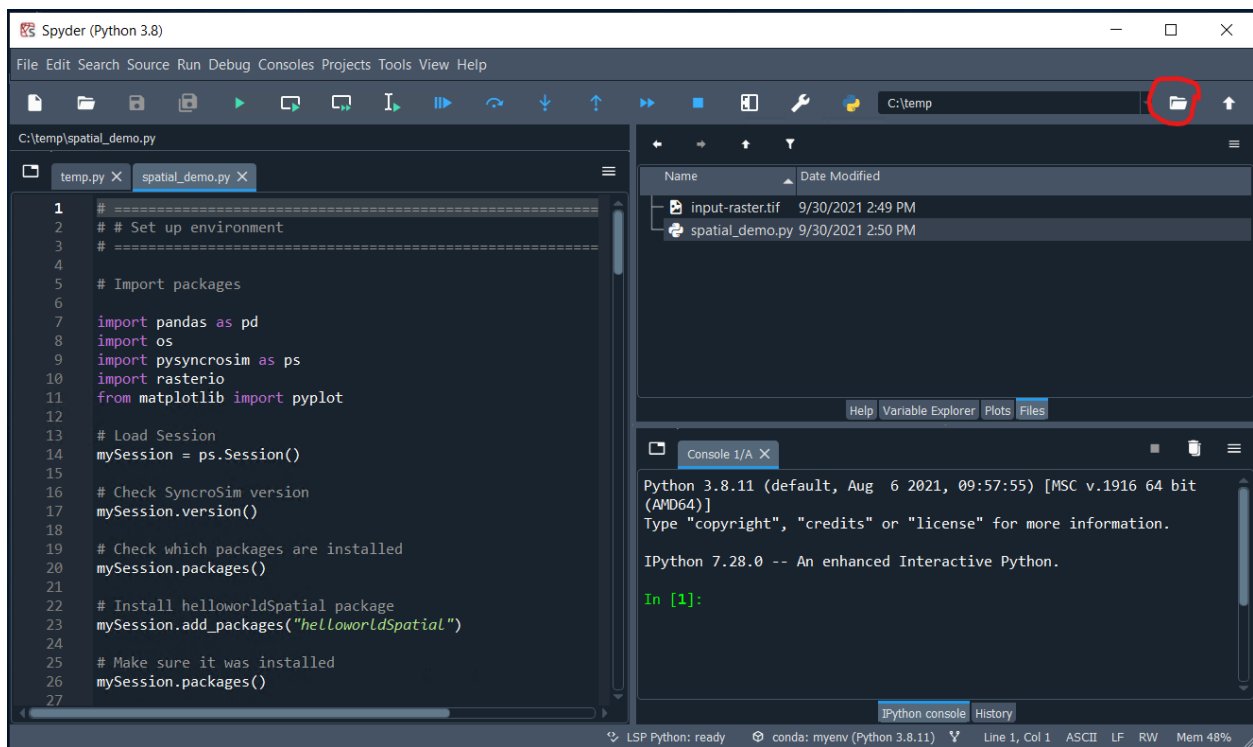
```
# Install spyder  
conda install spyder
```

2. Open the IDE by typing spyder in the command prompt.

```
# Open spyder  
spyder
```

Note: you may get a pop-up saying you have a missing dependency, `rtree`. You can safely ignore this warning.

3. Within the IDE, change the working directory to the directory containing your pysyncrosim scripts (e.g. `spatial_demo.py` and `input-raster.tif`)
4. Open and run your pysyncrosim scripts from the left-hand window. You can run scripts line-by-line in spyder by selecting the line(s) you want to run and pressing F9.



API REFERENCE

<code>environment</code>	
<code>helper</code>	
<code>library.Library</code>	A class to represent a SyncroSim Library.
<code>project.Project</code>	A class to represent a SyncroSim Project.
<code>raster.Raster</code>	A class to represent a raster object.
<code>scenario.Scenario</code>	A class representing a SyncroSim Scenario.
<code>session.Session</code>	A class to represent a SyncroSim Session.

4.1 pysyncrosim.environment

Functions

<code>progress_bar([report_type, iteration, ...])</code>	Begins, steps, ends, and reports progress for a SyncroSim simulation.
<code>runtime_input_folder(scenario, datasheet_name)</code>	Creates a SyncroSim Datasheet input folder.
<code>runtime_output_folder(scenario, datasheet_name)</code>	Creates a SyncroSim Datasheet output folder.
<code>runtime_temp_folder(folder_name)</code>	Creates a SyncroSim Datasheet temporary folder.
<code>update_run_log(*message[, sep])</code>	Begins, steps, ends, and reports progress for a SyncroSim simulation.

4.2 pysyncrosim.helper

Functions

<code>library(name[, session, package, addons, ...])</code>	Creates a new SyncroSim Library and opens it as a Library class instance.
---	---

4.3 pysyncrosim.library.Library

class pysyncrosim.library.**Library**(*location, session*)

A class to represent a SyncroSim Library.

__init__(*location, session*)

Initializes a pysyncrosim Library instance.

Parameters

- **location** (*String*) – Filepath to Library location on disk.
- **session** (*Session*) – pysyncrosim Session instance.

Return type

None.

Methods

<code>__init__(location, session)</code>	Initializes a pysyncrosim Library instance.
<code>backup()</code>	Creates a backup of a SyncroSim Library.
<code>datasheets([name, summary, optional, empty, ...])</code>	Retrieves a DataFrame of Library Datasheets.
<code>delete([project, scenario, force])</code>	Deletes a SyncroSim class instance.
<code>disable_addons(name)</code>	Disable addon package(s) of a SyncroSim Library.
<code>enable_addons(name)</code>	Enable addon package(s) of a SyncroSim Library.
<code>projects([name, pid, summary, overwrite])</code>	Creates or opens one or more SyncroSim Projects in the Library.
<code>run([scenarios, project, jobs, ...])</code>	Runs a list of Scenario objects.
<code>save_datsheet(name, data[, scope])</code>	Saves a pandas DataFrame as a SyncroSim Datasheet.
<code>scenarios([name, project, sid, pid, ...])</code>	Retrieves a Scenario or DataFrame of Scenarios in this Library.
<code>update()</code>	Updates a SyncroSim Library.

Attributes

<code>addons</code>	Retrieves the addon(s) this Library is using.
<code>date_modified</code>	Retrieves the last date this Library was modified.
<code>description</code>	Gets or sets the description for this Library.
<code>info</code>	Retrieves information about this Library.
<code>location</code>	Retrieves the file path to this Library.
<code>name</code>	Retrieves or sets the name of this Library.
<code>owner</code>	Gets or sets the owner of this Library.
<code>package</code>	Retrieves the package this Library is using.
<code>readonly</code>	Gets or sets the read-only status of this Library.
<code>session</code>	Retrieves the Session associated with this Library.

4.4 pysyncrosim.project.Project

class pysyncrosim.project.**Project**(*pid, name, library*)

A class to represent a SyncroSim Project.

__init__(*pid, name, library*)

Initializes a pysyncrosim Project instance.

Parameters

- **pid** (*Int*) – Project ID.
- **name** (*String*) – Project name.
- **library** (*Library*) – pysyncrosim Library instance.

Return type

None.

Methods

<code>__init__(pid, name, library)</code>	Initializes a pysyncrosim Project instance.
<code>copy([name])</code>	Creates a copy of an existing Project.
<code>datasheets([name, summary, optional, empty, ...])</code>	Retrieves a DataFrame of Project Datasheets.
<code>delete([scenario, force])</code>	Deletes a Project or Scenario.
<code>run([scenarios, jobs, copy_external_inputs])</code>	Runs a list of Scenario objects.
<code>save_datasheet(name, data)</code>	Saves a Project-scoped Datasheet.
<code>scenarios([name, sid, optional, summary, ...])</code>	Retrieve a DataFrame of Scenarios in this Project.

Attributes

<code>date_modified</code>	Gets the last date this Project was modified.
<code>description</code>	Gets or sets the Project description.
<code>info</code>	Retrieves Project information.
<code>library</code>	Retrieves the Library the Project belongs to.
<code>name</code>	Retrieves or sets Project name.
<code>owner</code>	Gets or sets the owner of this Project.
<code>pid</code>	Retrieves Project ID.
<code>readonly</code>	Gets or sets the read-only status for this Project.

4.5 pysyncrosim.raster.Raster

class pysyncrosim.raster.**Raster**(*source, iteration=None, timestep=None*)

A class to represent a raster object.

__init__(*source, iteration=None, timestep=None*)

Methods

<code>__init__(source[, iteration, timestep])</code>	
<code>values([band])</code>	Gets the values in each cell of the raster

Attributes

<code>crs</code>	Gets the coordinate system of the raster
<code>dimensions</code>	Gets the dimensions of the raster
<code>extent</code>	Gets the extent of the raster
<code>name</code>	Gets the name of the raster
<code>resolution</code>	Gets the resolution of the raster
<code>source</code>	Gets the filepath of the raster

4.6 pysyncrosim.scenario.Scenario

class `pysyncrosim.scenario.Scenario`(*sid=None, name=None, project=None, library=None*)

A class representing a SyncroSim Scenario.

`__init__(sid=None, name=None, project=None, library=None)`

Methods

<code>__init__([sid, name, project, library])</code>	
<code>copy([name])</code>	Creates a copy of an existing Scenario class instance.
<code>datasheet_rasters(datasheet[, column, ...])</code>	Retrieves spatial data columns from one or more SyncroSim Datasheets.
<code>datasheets([name, summary, optional, empty, ...])</code>	Retrieves a DataFrame of Scenario Datasheets.
<code>delete([force])</code>	Deletes a Scenario.
<code>dependencies([dependency, remove, force])</code>	Gets, sets, or removes dependencies from a Scenario.
<code>ignore_dependencies([value])</code>	Retrieves or sets the Datafeeds to ignore for a Scenario.
<code>merge_dependencies([value])</code>	Retrieves or sets whether or not a Scenario is configured to merge dependencies at run time.
<code>results([sid])</code>	Retrieves DataFrame of Results Scenarios or retrieves a Results Scenario instance for this Scenario.
<code>run([jobs, copy_external_inputs])</code>	Runs a Scenario.
<code>run_log()</code>	Returns a run log for a Results Scenario.
<code>save_datasheet(name, data)</code>	Saves a pandas DataFrame as a SyncroSim Datasheet.

Attributes

<code>date_modified</code>	Gets the last date this Scenario was modified.
<code>description</code>	Gets or sets the Scenario description.
<code>info</code>	Retrieves Scenario information.
<code>is_result</code>	Retrieves information about whether this Scenario is a Results Scenario or not.
<code>library</code>	Retrieves Library class instance associated with this Library.
<code>name</code>	Retrieves or sets a Scenario name.
<code>owner</code>	Gets or sets the owner of this Scenario.
<code>parent_id</code>	Retrieves the ID of the parent Scenario of a Results Scenario.
<code>project</code>	Retrieves Project class instance associated with this Scenario.
<code>project_id</code>	Gets the Project ID associated with this Scenario.
<code>readonly</code>	Gets or sets the read-only status for this Scenario.
<code>sid</code>	Retrieves Scenario ID.

4.7 pysyncrosim.session.Session

class `pysyncrosim.session.Session`(*location=None, silent=True, print_cmd=False*)

A class to represent a SyncroSim Session.

__init__(*location=None, silent=True, print_cmd=False*)

Initializes a pysyncrosim Session instance.

Parameters

- **location** (*Str, optional*) – Filepath to SyncroSim executable. If None, then uses default location on windows. The default is None.
- **silent** (*Logical, optional*) – If True, will not print warnings from the console. The default is True.
- **print_cmd** (*Logical, optional*) – If True, arguments from the console command will be printed. The default is False.

Raises

- **ValueError** – Raises error if the location given does not exist.
- **RuntimeError** – Raises error if the version of the SyncroSim installation is incompatible with the current version of pysyncrosim.

Return type

None.

Methods

<code>__init__([location, silent, print_cmd])</code>	Initializes a pysyncrosim Session instance.
<code>add_packages(packages)</code>	Installs a package.
<code>packages([installed])</code>	Retrieves DataFrame of installed packages.
<code>remove_packages(packages)</code>	Uninstalls a package.
<code>update_packages([packages])</code>	Updates a package to the newest version.
<code>version()</code>	Retrieves SyncroSim version.

Attributes

<code>location</code>	Retrieves the location for this Session.
<code>print_cmd</code>	Gets or sets the print_cmd status of the SyncroSim Session.
<code>silent</code>	Gets or sets the silent status for this Session.

SUPPORT

To report a bug, ask a question, or request an enhancement for the pysyncrosim software, fill out an [issue](#) on the [pysyncrosim GitHub](#) or send an email to info@apexrms.com.

INDICES AND TABLES

- genindex
- modindex

Symbols

`__init__()` (*pysyncrosim.library.Library* method), 24
`__init__()` (*pysyncrosim.project.Project* method), 25
`__init__()` (*pysyncrosim.raster.Raster* method), 25
`__init__()` (*pysyncrosim.scenario.Scenario* method),
26
`__init__()` (*pysyncrosim.session.Session* method), 27

L

Library (class in *pysyncrosim.library*), 24

M

module

pysyncrosim.environment, 23
pysyncrosim.helper, 23

P

Project (class in *pysyncrosim.project*), 25

pysyncrosim.environment

module, 23

pysyncrosim.helper

module, 23

R

Raster (class in *pysyncrosim.raster*), 25

S

Scenario (class in *pysyncrosim.scenario*), 26

Session (class in *pysyncrosim.session*), 27