Octeract Engine

Python API Reference Manual

January 17, 2020

Page intentionally left blank.

Contents

Python API Reference Manual
Octeract Engine Beta v1.01.17

January 17, 2020

1 Overview

The Octeract Python API is a package that allows the Octeract Engine library functions to be accessed from within the Python programming language. One can use the API interactively via the Octeract Shell (a wrapper around IPython3) or a standard Python interperter. When first familiarising yourself with the API, we recommend using the Octeract Shell interactively, as the tab completion functionality provided is extremely useful when browsing through the available API calls. The API can also be utilised for writing either small stand-alone scripts or Python libraries within larger scale applications.

This manual assumes that you are familiar with the Python programming language, and that you have the appropriate version of the Python distribution installed on your system. The API itself requires Python version 3.6 or above, and the Octeract Shell has a dependency on IPython 3. An issue often encountered is trying to load the API library using a Python 2 interpreter, which will fail. To alleviate this, we suggest explicitly running your scripts with the Python 3 interpreter (e.g. `python3 myScript.py`, rather than `python myScript.py`). If you have installed the Octeract Engine via Docker, then all dependencies will be taken care of by simply running your python scripts through the Octeract Shell.

The standard workflow for solving a global optimization problem using the API involves working with the Octeract *Model* object. An optimization problem can be built by using *Model* functions to add/remove variables, set the objective, and add/remove constraints. Once the *Model* has been built, the `global_solve()` function can be invoked, which sends the problem to the Octeract engine. As soon as the problem is solved to global optimality, a solution file is written, automatically read by the API and stored as an object in memory. The properties of the solution object (e.g. the optimal objective value and values of the decision variables) can then be accessed by the user and manipulated or fed in as data as part of a larger program.

The best way to familiarise yourself with the API is to read the Getting Started guide in the **User Manual** or website (**https://www.octeract.com/getting-started**), and to work through some of the simple examples provided in `app/octeract/examples/python/`. You can run the examples in the folder via the Octeract Shell, via the `run` command in IPython, e.g. `%run -i "app/octeract/examples/python/example.py"` (see Section 5).

2 Installation of Octeract Engine for Linux and macOS

Getting started guide for Octeract Engine Beta.

2.1 Prerequisites

1. Internet connectivity is required throughout the procedure.
2. Download and install **Docker** for your OS:
   - Mac: **https://docs.docker.com/docker-for-mac/install/**
   - Linux: **https://docs.docker.com/install/linux/docker-ce/ubuntu/**
3. Download and install Python 3 for your OS from **https://www.python.org/downloads/**
4. The rest of this section assumes familiarity with shell environments.

2.2 Installation using cross-platform Python script

2.2.1 Installation

In this section instructions are provided for the installation of Octeract Engine using the cross-platform Python script.

1. Download the `octeract-installer.zip` and `octeract-engine.tgz` files from the Downloads page **https://octeract.com/downloads/**
2. Extract (unzip) the contents of the `octeract-installer.zip` file to a selected location in your PC.
3. In a terminal window, navigate to the folder "installer" and run

   ```
   python3 octeract_installer.py
   ```

   from the location where the `octeract_installer.py` has been saved and follow the on-screen instructions.

   Note

   During installation, if you are unsure about the tmp and/or mount directory you are advised to use the defaults.

4. You are advised to append your system `PATH` so that the octeract engine can be invoked throughout your system.

   Note

   The installation reports the installation directory upon completion which will be referred to as `SAVE_LOCATION` throughout this section.

2.2.2 Starting and stopping the Octeract Engine Container

To start/stop the Octeract Engine container you can run:

```
octeract_launcher.py
```

if you chose to append your system `PATH` or

```
python3 SAVE_LOCATION/octeract-engine/octeract_launcher.py
```

if you chose not to append your system `PATH`.

2.2.3 Uninstalling the Octeract Engine

You can simply uninstall the Octeract Engine by running the command:

```
python3 octeract_installer.py --uninstall
```

3 Installation of Octeract Engine for Windows 10

Getting started guide for Octeract Engine Beta cross-platform edition

3.1 Prerequisites

1. Internet connectivity is required throughout the procedure.
2. Only Windows 10 **Pro** and **Enterprise** editions are currently supported.
3. Download and install **Docker** for Windows from **https://docs.docker.com/docker-for-windows/install/**
4. Download and install Python 3 for your OS from **https://www.python.org/downloads/**
5. Download Octeract DGO Engine Docker archive from **https://www.octeract.com/downloads/**
6. The rest of this section assumes some familiarity with Powershell.

3.2 Installation using the Graphical User Interface (GUI)

3.2.1 Installation

In this section the user is guided through the installation of the Octeract Engine Beta cross-platform edition using the Graphical User Interface (GUI).

1. Download the `octeract-installer.zip` and `octeract-engine.tgz` files from the Downloads page **https://octeract.com/downloads/**
2. Extract (unzip) the contents of the `octeract-installer.zip` file to a selected location in your PC.
3. Open the folder "installer" and double click `octeract_installer.py` to launch the installer wizard.
4. Read and accept the license agreement by clicking "I Agree".
5. If Docker is already installed and running check the "I have installed and started Docker", otherwise please see section 3.1 and start Docker before you proceed to the next step.
6. Click Browse and locate the file "octeract-engine.tgz" that was downloaded on your PC. Click Next.
7. Click Browse and select the installation folder of your choice. Click Next.
8. (Optional) Check the "Add to `PATH`" box before clicking Next if you want to automatically update your `PATH` variables to include the installation location.
9. Click Browse and choose the mount directory that the Octeract Engine Docker will have access to. To choose the default location, just click "Select Folder" without explicitly selecting a folder. Click Next.
10. Click Browse and choose the temporary directory that the Octeract Engine Docker will have access to. To choose the default location, just click "Select Folder" without explicitly selecting a folder. Click Next.

Note

If you are unsure about the temporary and/or mount directory you are advised to use `C:\\Users\YOUR_USERNAME\AppData\Local\Temp` and `C:\\Users\YOUR_USERNAME` respectively.

11. Wait for the installation to complete and click Next.
12. Octeract Engine has been successfully installed. Click Done and close the installation wizard

3.2.2 Starting and stopping the Octeract Engine Container

To start/stop the Octeract Engine container you can follow the steps:

1. Navigate to the location where the folder "octeract-engine" was installed.
2. Double click the `octeract_launcher.py` file. This will launch a PowerShell session.
3. Choose and type the appropriate number and hit Enter. Docker may request your user password to run.

3.2.3 Uninstalling the Octeract Engine

You can simply uninstall the Octeract Engine by running the command in a PowerShell terminal:

```
py octeract_installer.py --uninstall
```

4 A Simple Example

In this section, we will go through building a simple optimization problem based on the Haverly Pooling problem (a quadratically constrained problem). The current version of the API primarily uses a string-based interface, meaning the majority of the `set()` and `add()` methods will take strings as arguments. Future versions will have operator overloaded expression type objects that the user can manipulate directly without having to use quotation marks. The rest of this section assumes the directory containing `octeract_shell.py` is in your system PATH.

$$
\begin{aligned}
\text{minimize} \quad & x_1 - x_2 \\
\text{subject to} \quad & x_1 - 6x_3 - 16x_4 - 10x_5 = 0 \\
& x_2 - 9x_6 - 15x_7 = 0 \\
& x_6 - x_8 - x_10 = 0 \\
& x_7 - x_9 - x_11 = 0 \\
& x_3 + x_4 - x_10 - x_11 = 0 \\
& x_5 - x_8 - x_9 = 0 \\
& x_{12}(x_{10} + x_{11}) - 3x_2 - x_4 = 0 \\
& x_{12}x_{10} - 2.5x_{10} - 0.5x_8 <= 0 \\
& x_{12}x_{11} - 1.5x_{11} + 0.5x_9 <= 0
\end{aligned}
$$

1. Launch the Octeract Shell

```
octeract-shell.py
```

2. Instantiate a new model

```
In [1]: m = Model()
```

3. Set a linear objective

```
In [2]: m.set_objective("x1-x2")
```

Note that you do not need to explicitly add the variables x1 and x2 to the model - the API will automatically add the variables, and you can set their properties (such as domain bounds and type) later on. The sense of the objective defaults to minimize.

4. Add the constraints to your model

```
In [3]: m.add_constraint("x1 - 6*x3 - 16*x4 - 10*x5 = 0")
In [4]: m.add_constraint("x2 - 9*x6 - 15*x7 = 0")
In [5]: m.add_constraint("x6 - x8 - x10 = 0")
In [6]: m.add_constraint("x7 - x9 - x11 = 0")
In [7]: m.add_constraint("x3 + x4 - x10 - x11 = 0")
In [8]: m.add_constraint("x5 - x8 - x9 = 0")
In [9]: m.add_constraint("x12*(x10 + x11) - 3*x3 - x4 = 0")
In [10]: m.add_constraint("x12*x10 - 2.5*x10 - 0.5*x8 <= 0")
In [11]: m.add_constraint("x12*x11 - 1.5*x11 + 0.5*x9 <= 0")
```

5. Set the variable bounds

```
In [12]: m.set_variable_lb("x1", 0)
In [13]: m.set_variable_lb("x2", 0)
In [14]: m.set_variable_lb("x3", 0)
In [15]: m.set_variable_lb("x4", 0)
In [16]: m.set_variable_lb("x5", 0)
In [17]: m.set_variable_bounds("x6", 0, 100)
In [18]: m.set_variable_bounds("x7", 0, 200)
In [19]: m.set_variable_lb("x8", 0)
In [20]: m.set_variable_lb("x9", 0)
In [21]: m.set_variable_lb("x10", 0)
In [22]: m.set_variable_lb("x11", 0)
In [23]: m.set_variable_lb("x12", 0)
```

6. At this point, the model is built and ready to be sent to the solver. However, before doing so, let's check that everything looks correct by printing the model.

```
In [24]: print(m)
```

7. Now we're ready to solve the problem to global optimality. Invoke the `global_solve()` command to send your model to the Octeract Engine.

```
In [25]: m.global_solve()
```

8. The solver will produce it's own real-time output so you will know when the solving process is done and the global solution is found. Finally, you can print the objective value

```
In [26]: m.get_solution_objective_value()
Out[26]: -399.9999999999204
```

   which is -400 in this case, and the expected global solution. Well done, you have just solved your first nonconvex problem to global optimality using the Octeract Python API!

The majority of the available methods have intuitive names, and you can use the tab completion functionality of the Octeract Shell, or your IDE, to see a list of the available methods. Since Python is a very flexible language, users will easily be able to integrate their code as part of a larger overall project.

5 Running Python scripts via the Octeract Shell

In many cases, especially for larger projects, you will probably want to write your Python scripts outside of the interactive environment. In this case, you can still execute your scripts via the Octeract Shell, as outlined in the steps below. It is assumed that the directory containing `octeract_shell.py` is in your system PATH.

1. Launch the Octeract Shell

```
octeract-shell.py
```

2. If you have written a Python script called myScript.py Use the `run` command to execute your Python script.

```
In [1]: %run -i "myScript.py"
```

   You should see all the relevant output (if any) in your console window.

6 API Reference

This section provides a comprehensive list of all the API methods. Each subsection contains a brief description of the method, the arguments the function takes as input, the Return type, and example usages.

6.1 Model

`Model()`

Description:
Instantiates an empty model.

Arguments:
None

Return type:
None

Example usage:
```
m = Model()
```

`Model.add_constraint()`

add_constraint(constraint_string)

Description:
Adds a constraint to the model.

Arguments:
constraint_string: the function string of the constraint to be added

Return type:
None

Example usage:
```
model.add_constraint("x1 + sin(x2*x3) = 2")
```

`Model.add_variable()`

add_variable(variable_name, lb=-OCT_INFTY, ub=OCT_INFTY, variable_type=CONT)

Description:
Adds a variable to the model

Arguments:
variable_name: the name of the variable to be added
lb: the lower bound of the variable
ub: the upper bound of the variable
variable_type: the variable type (CONT or BIN)

Return type:
None

Example usage:

```
model.add_variable("x1")
model.add_variable("x1", -1)
model.add_variable("x1", -10, 10)
model.add_variable("x1", 0, 1, BIN)
```

`Model.classify_problem()`

classify_problem()

Description:
Detects the problem structure (MINLP, NLP, MIQCQP, QCQP, MIQP, QP, MILP, LP) of the current model.

Arguments:
None

Return type:
String

Example usage:

```
model.classify_problem()
```

`Model.clear()`

clear()

Description:
Clears the model - removes the objective, all variables, all constraints and any settings the user might have specified.

Arguments:
None

Return type:
None

Example usage:

```
model.clear()
```

`Model.detect_bounds()`

detect_bounds()

Description:
Tries to detect valid variable bounds based on the constraints present in the model.

Arguments:
None

Return type:

None

Example usage:
`model.detect_bounds()`

`Model.detect_problem_convexity()`

detect_problem_convexity()

Description:
Tries to determine whether the current model is convex using various techniques. If the problem is proven to be convex, the function returns **True**, otherwise it returns **False**.

Arguments:
None

Return type:
Boolean

Example usage:
`model.detect_problem_convexity()`

`Model.get_binary_constraint_names()`

get_binary_constraint_names()

Description:
Returns a list of constraint names which only contain binary variables.

Arguments:
None

Return type:
A list of constraint names

Example usage:
`model.get_binary_constraint_names()`

`Model.get_binary_variable_names()`

get_binary_variable_names()

Description:
Returns the binary variables of the model.

Arguments:
None

Return type:
A list of variable names

Example usage:

```
model.get_binary_variable_names()
```

Model.get_bound_multipliers()

get_bound_multipliers()

Description:
Returns a map of the decision variable names to their bound multipliers at the optimal solution.

Arguments:
None

Return type:
Dictionary from String to Double

Example usage:
```
model.get_bound_multipliers()
```

Model.get_constraint_lb()

get_constraint_lb(constraint_name)

Description:
Returns the lower bound of a constraint.

Arguments:
constraint_name: the name of the constraint

Return type:
Double

Example usage:
```
model.get_constraint_lb("c1")
```

Model.get_constraint_lhs()

get_constraint_lhs(constraint_name)

Description:
Returns the function string (left hand side) of a constraint.

Arguments:
constraint_name: the name of the constraint

Return type:
String

Example usage:
```
model.get_constraint_lhs("c1")
```

`Model.get_constraint_multipliers()`

get_constraint_multipliers()

Description:
Returns a map of the constraint names to their Lagrange multipliers at the optimal solution.

Arguments:
None

Return type:
Dictionary from String to Double

Example usage:
```
model.get_constraint_multipliers()
```

`Model.get_constraint_names()`

get_constraint_names()

Description:
Returns all the constraints of the model.

Arguments:
None

Return type:
A list of constraint names

Example usage:
```
model.get_constraint_names()
```

`Model.get_constraint_ub()`

get_constraint_ub(constraint_name)

Description:
Returns the upper bound of a constraint.

Arguments:
constraint_name: the name of the constraint

Return type:
Double

Example usage:
```
model.get_constraint_ub("c1")
```

`Model.get_constraint_variables()`

get_constraint_variables(constraint_name)

Description:
Returns all the variables of a constraint.

Arguments:
constraint_name: the name of the constraint

Return type:
A list of variable names

Example usage:
```
model.get_constraint_variables("c1")
```

Model.get_continuous_variable_names()

get_continuous_variable_names()

Description:
Returns the continuous variables of the model.

Arguments:
None

Return type:
A list of variable names

Example usage:
```
model.get_continuous_variable_names()
```

Model.get_linear_constraint_names()

get_linear_constraint_names()

Description:
Returns the linear constraints of the model.

Arguments:
None

Return type:
A list of constraint names

Example usage:
```
model.get_linear_constraint_names()
```

Model.get_linear_equality_constraint_names()

get_linear_equality_constraint_names()

Description:
Returns the linear equality constraints of the model.

Arguments:

None

Return type:
A list of constraint names

Example usage:
```
model.get_linear_equality_constraint_names()
```

`Model.get_linear_variable_names()`

get_linear_variable_names()

Description:
Returns the linear variables of the model.

Arguments:
None

Return type:
A list of variable names

Example usage:
```
model.get_linear_variable_names()
```

`Model.get_nonlinear_constraint_names()`

get_nonlinear_constraint_names()

Description:
Returns the nonlinear constraints of the model.

Arguments:
None

Return type:
A list of constraint names

Example usage:
```
model.get_nonlinear_constraint_names()
```

`Model.get_nonlinear_equality_constraint_names()`

get_nonlinear_equality_constraint_names()

Description:
Returns the nonlinear equality constraints of the model.

Arguments:
None

Return type:

A list of constraint names

Example usage:
`model.get_nonlinear_equality_constraint_names()`

`Model.get_nonlinear_variable_names()`

get_nonlinear_variable_names()

Description:
Returns the nonlinear variables of the model.

Arguments:
None

Return type:
A list of variable names

Example usage:
`model.get_continuous_nonlinear_names()`

`Model.get_objective_function_string()`

get_objective_function_string()

Description:
Returns the function string of objective.

Arguments:
None

Return type:
String

Example usage:
`model.get_objective_function_string()`

`Model.get_problem_type()`

get_problem_type()

Description:
Returns the problem structure (MINLP, NLP, MIQCQP, QCQP, MIQP, QP, MILP, LP).

Arguments:
None

Return type:
String

Example usage:
`model.get_problem_type()`

`Model.get_solution_objective_value()`

get_solution_objective_value()

Description:
Returns the value of the Objective at the optimal solution.

Arguments:
None

Return type:
Double

Example usage:
```
model.get_solution_objective_value()
```

`Model.get_solution_path()`

get_solution_path()

Description:
Returns the path where the solution file will be written to.

Arguments:
None

Return type:
String

Example usage:
```
model.get_solution_path()
```

`Model.get_solution_vector()`

get_solution_vector()

Description:
Returns a map of the decision variable names to their optimal solution values.

Arguments:
None

Return type:
Dictionary from String to Double

Example usage:
```
model.get_solution_vector()
```

`Model.get_variable_lb()`

get_variable_lb(variable_name)

Description:
Returns the lower bound of a variable.

Arguments:
variable_name: the name of the variable

Return type:
Double

Example usage:
```
model.get_variable_lb("x1")
```

Model.get_variable_names()

get_variable_names()

Description:
Returns all the variables of the model.

Arguments:
None

Return type:
A list of variable names

Example usage:
```
model.get_variable_names()
```

Model.get_variable_ub()

get_variable_ub(variable_name)

Description:
Returns the upper bound of a variable.

Arguments:
variable_name: the name of the variable

Return type:
Double

Example usage:
```
model.get_variable_ub("x1")
```

Model.global_solve()

global_solve()

Description:
Solves the model to global optimality.

Arguments:

None

Return type:
None

Example usage:
```
model.global_solve()
```

Model.import_model_file()

import_model_file(problem_file_path)

Description:
Imports a model file, which can be either in ASL format (.nl extension) or AMPL format (.mod extension).

Arguments:
problem_file_path: the path to the problem file

Return type:
None

Example usage:
```
model.import_model_file("/home/me/myFolder/myProblem.nl")
model.import_model_file("/home/me/myFolder/myProblem.mod")
```

Model.import_solution_file()

import_solution_file(solution_file_path)

Description:
Imports an Octeract solution file (.octsol extension) and stores the solution data in memory.

Arguments:
problem_file_path: the path to the problem file

Return type:
None

Example usage:
```
model.import_solution_file("/home/me/mySolutionsFolder/myProblem.octsol")
```

Model.is_problem_convex()

is_problem_convex()

Description:
Returns whether the current model has been detected to be convex.

Arguments:
None

Return type:
Boolean

Example usage:
`model.is_problem_convex()`

`Model.local_solve()`

local_solve()

Description:
Solves the model to local optimality.

Arguments:
None

Return type:
None

Example usage:
`model.local_solve()`

`Model.multistart_local_solve()`

multistart_local_solve()

Description:
Solves the model to local optimality using multistart.

Arguments:
None

Return type:
None

Example usage:
`model.multistart_local_solve()`

`Model.num_binary_constraints()`

num_binary_constraints()

Description:
Returns the number of constraints which only contain binary variables.

Arguments:
None

Return type:
Integer

Example usage:

```
model.num_binary_constraints()
```

Model.num_binary_variables()

num_binary_variables()

Description:
Returns the number of binary variables present in the model.

Arguments:
None

Return type:
Integer

Example usage:
```
model.num_binary_variables()
```

Model.num_constraints()

num_constraints()

Description:
Returns the total number of constraints present in the model.

Arguments:
None

Return type:
Integer

Example usage:
```
model.num_constraints()
```

Model.num_continuous_variables()

num_continuous_variables()

Description:
Returns the number of continuous variables present in the model.

Arguments:
None

Return type:
Integer

Example usage:
```
model.num_continuous_variables()
```

Model.num_linear_constraints()

num_linear_constraints()

Description:
Returns the number of linear constraints present in the model.

Arguments:
None

Return type:
Integer

Example usage:
```
model.num_linear_constraints()
```

Model.num_nonlinear_constraints()

num_nonlinear_constraints()

Description:
Returns the number of nonlinear constraints present in the model.

Arguments:
None

Return type:
Integer

Example usage:
```
model.num_nonlinear_constraints()
```

Model.num_nonlinear_equality_constraints()

num_nonlinear_equality_constraints()

Description:
Returns the number of nonlinear equality constraints present in the model.

Arguments:
None

Return type:
Integer

Example usage:
```
model.num_nonlinear_equality_constraints()
```

Model.num_nonlinear_variables()

num_nonlinear_variables()

Description:
Returns the number of nonlinear variables present in the model.

Arguments:
None

Return type:
Integer

Example usage:
```
model.num_nonlinear_variables()
```

Model.num_variables()

num_variables()

Description:
Returns the total number of variables present in the model.

Arguments:
None

Return type:
Integer

Example usage:
```
model.print_problem_summary()
```

Model.print_problem_summary()

print_problem_summary()

Description:
Prints a summary of the loaded or constructed optimization problem onto the console.

Arguments:
None

Return type:
None

Example usage:
```
model.print_solution_summary()
model.print_solution_summary()
```

Model.print_solution_summary()

print_solution_summary()

Description:
Prints a summary of the solution obtained from a local_solve or global_solve call.

Arguments:
None

Return type:
None

Example usage:
```
model.print_solution_summary()
```

Model.read_options_file()

read_options_file(options_file_path)

Description:
Loads an options file, which will be read by the solver.

Arguments:
options_file_path: the path to the options file

Return type:
None

Example usage:
```
model.read_options_file("/home/me/myFolder/octeract.opt")
```

Model.remove_all_constraints()

remove_all_constraints()

Description:
Removes all constraints from the model.

Arguments:
None

Return type:
None

Example usage:
```
model.remove_all_constraints()
```

Model.remove_all_variables()

remove_all_variables()

Description:
Removes all variables from the model. This effectively clears the model, as no objective or constraints can exist in the model without the presence of variables.

Arguments:
None

Return type:
None

Example usage:
```
model.remove_all_variables()
```

Model.remove_constraint()

remove_constraint(constraint_name)

Description:
Removes a constraint of the model.

Arguments:
constraint_name: the name of the constraint to be removed

Return type:
None

Example usage:
```
model.remove_constraint("c1")
```

Model.remove_objective()

remove_objective()

Description:
Removes the objective of the model

Arguments:
None

Return type:
None

Example usage:
```
model.remove_objective()
```

Model.remove_variable()

remove_variable(variable_name)

Description:
Removes a variable of the model.

Arguments:
variable_name: the name of the variable to be removed

Return type:
None

Example usage:

```
model.remove_variable("x1")
```

Model.set_objective()

set_objective(objective_string, sense=MINIMIZE)

Description:
Sets the objective of the model

Arguments:
objective_string: the function string of the objective
sense: the objective sense (MINIMIZE or MAXIMIZE)

Return type:
None

Example usage:
```
model.set_objective("x1*log(x1) + x1*x2")
model.set_objective("x1*x2 + log(x3)", MAXIMIZE)
```

Model.set_solver_timeout()

set_solver_timeout(timeout)

Description:
Sets a timeout for the solver.

Arguments:
timeout: the timeout in seconds

Return type:
None

Example usage:
```
model.set_solver_timeout(10)
```

Model.set_variable_bounds()

set_variable_bounds(variable_name, lb, ub)

Description:
Sets the bounds of a variable.

Arguments:
variable_name: the name of the variable
lb: the lower bound of the variable
ub: the upper bound of the variable

Return type:
None

Example usage:

```
model.set_variable_bounds("x1", -10, 10)
```

Model.set_variable_lb()

set_variable_lb(variable_name, lb)

Description:
Sets the lower bound of a variable.

Arguments:
variable_name: the name of the variable
lb: the lower bound of the variable

Return type:
None

Example usage:
```
model.set_variable_lb("x1", 2)
```

Model.set_variable_type()

set_variable_type(variable_name, type)

Description:
Sets the type of a variable.

Arguments:
variable_name: the name of the variable
type: the type of the variable

Return type:
None

Example usage:
```
model.set_variable_type("x1", CONT)
model.set_variable_type("x1", BIN)
```

Model.set_variable_ub()

set_variable_ub(variable_name, ub)

Description:
Sets the upper bound of a variable.

Arguments:
variable_name: the name of the variable
ub: the upper bound of the variable

Return type:
None

Example usage:

```
model.set_variable_ub("x1", 100)
```

Model.write_current_solution_to_file()

write_current_solution_to_file(file_path)

Description:
Writes the current solution (if any) to a file (Octeract solution format with a .octsol extension).

Arguments:
file_path: the path of the solution file

Return type:
None

Example usage:
```
model.write_current_solution_to_file("/home/me/mySolutions/myProblem.octsol")
```

Model.write_problem_to_mod_file()

write_problem_to_mod_file(file_path)

Description:
Writes the current model to an AMPL (extension .mod) file.

Arguments:
file_path: the path of the .mod file

Return type:
None

Example usage:
```
model.write_problem_to_mod_file("/home/me/myProblems/myProblem.mod")
```

Model.write_problem_to_NL_file()

write_problem_to_NL_file(file_path)

Description:
Writes the current model to an ASL (extension .nl) file.

Arguments:
file_path: the path of the .nl file

Return type:
None

Example usage:
```
model.write_problem_to_NL_file("/home/me/myProblems/myProblem.nl")
```