Using the Lingo Python API

The Lingo Python interface allows you to send a model and it's supporting data to Lingo for solution. The interface can also return solution values to your Python code.

To learn more about the Lingo modeling language, see the Lingo manual. In particular, Chapter 11, *Interfacing with Other Applications,* may prove helpful.

For a quick start, there are several downloadable examples here: https://github.com/lindosystems/lingoapi-python/tree/main/examples that illustrate calling the Lingo API from Python.

Installing the Lingo Python API

You will need to install the Lingo API as an add-in package to your version of Python. You can do this from the command line with pip as follows (user input in ***bold italics***):

```
C:\pywork> pip install lingo_api
Collecting lingo_api
  Using cached lingo_api-19.0.2-cp310-cp310-win_amd64.whl (16 kB)
Requirement already satisfied: numpy>=1.19 in
c:\python\python310\lib\site-packages (from lingo_api) (1.22.3)
Installing collected packages: lingo_api
Successfully installed lingo_api-19.0.2
```

You may then test your installation as follows:

```
C:\pytest> python -m lingo_test
Lingo API is Working.
```

To run one of the Python examples, unpack the Chess example from GitHub, then enter the following:

```
C:\pytest\Chess> python chess.py

Global optimum found!
Brand      Peanut       Cashew     Produce
=========================================
Pawn         721.1538    48.0769   769.2308
Knight         0.0000     0.0000     0.0000
Bishop         0.0000     0.0000     0.0000
King          28.8462   201.9231   230.7692
=========================================
Totals          750.0      250.0     1000.0
```

Code Sample

Below is a Python code fragment that illustrates creating a Lingo model object. This code was extracted from the Chess blending example downloadable from GitHub.

```
lngFile = "chess.lng"
SUPPLY = np.array( [750, 250])      # Total supply of each type
PRICE = np.array( [2,3,4,5])        # price that each brand charge
FORMULA = np.array( [[15,10, 6, 2],
                     [1, 6,10,14]])  # formula matrix
PRODUCE = np.zeros(len(PRICE))      # variables
STATUS = -1                         # Lingo status of model

# Pointers used in the model for passing data and solution values
pointerDict = {"Pointer1":NUT_COUNT,
               "Pointer2":BRAND_COUNT,
               "Pointer3":SUPPLY,
               "Pointer4":PRICE,
               "Pointer5":FORMULA,
               "Pointer6":PRODUCE,
               "Pointer7":STATUS
}

# Pass the model to the Lingo API
lingo_api.Model(lngFile, pointerDict)
```

Model

Model data is passed in a Python object called `lingo.api` using the `Model` method. The calling sequence for `Model` is: `lingo_api.Model(lngFile, pointerDict, logFile=None)`, where:

- **`lngFile`**: A string containing the path to the Lingo model file. The model must be saved in Lingo in LNG (text) format. LG4 (binary) format model files may not currently be passed to the Lingo API. The model's expressions must also be bracketed with a `MODEL:` statement at the start and an `END` statement at the end of the model (refer to any of the sample LNG model files to see the placement of these commands).
- **`pointerDict:`** Is a Python dictionary with declaration: `Dictionary(key, Model Data)`, where `key` must be a unique string, where using "PointerN" is helpful when going between the Python and Lingo scripts. The model data can be a NumPy array, list, int, float. Each element in the dictionary will be converted to a NumPy array of type `np.double` before being sent to Lingo making it the preferred type. These pointer arguments are used by the Lingo API to load your input data, as well as for writing out solutions.
- **`logFile`**: An optional string path to a logfile that will created by the API. The log file is useful for debugging. Whenever you experience problems calling the Lingo API, be sure to review the contents of this log file for any errors.

Solve

Once a `Model` object has been created, use the `solve` function to call the Lingo API and process the model.

```
lingo_api.solve(model)
```

Any NumPy arrays in the `pointerDict` that are modified by Lingo will also be modified in place in Python.

Getters

To get the model's data, there are two functions: `get_pointer(key)` returns a single pointer and `get_pointerDict()` to return the entire pointer dictionary. As an example:

```
price = model.get_pointer("Price_Pointer")
pointerDict = model.get_pointerDict()
```

To get the file path to the Lingo model file use `get_lngFile()`, and use `get_logFile()` to get the log file path, e.g.:

```
lngFN = model.get_lngFile()
```

Setters

To set the pointer dictionary use `set_pointerDict(pointerDict()`. Pointer keypairs can be added or modified by using `set_pointer(key, pointer)`:

```
pointerDict = {"Pointer1":NUT_COUNT,
               "Pointer2":BRAND_COUNT,
               "Pointer3":SUPPLY,
               "Pointer4":PRICE,
               "Pointer5":FORMULA,
               "Pointer6":PRODUCE
}
model.set_pointerDict(pointerDict)
STATUS = -1
model.set_pointer("Pointer7", STATUS)
```

To change the Lingo file to a different file path, use the function `set_lngFile(lngFile)`. To set or change the path of the log file use the function `set_logFile(logFile)`.

```
lngFile = "path/to/model.lng"
logFile = "path/to/modelLog.log"
model.set_lngFile(lngFile)
model.set_logFile(logFile)
```

Sending Data To and Receiving the Solution Back From Lingo

For sending and receiving data in the Lingo API, the `@POINTER(i)` statement is used. If data is being sent to Lingo, the `@POINTER()` statement is placed on the righthand-side: `SUPPLY = @POINTER(3)`. If data is being sent back to Python from Lingo, the `@POINTER()` statement is placed on the lefthand-side: `@POINTER(6) = PRODUCE`.

Note that to get the solution status of the model, you can return the value of the `@STATUS()` function to Lingo as was done in the *Code Sample* section above. Possible status conditions are:

@STATUS() Code	Interpretation
0	Global Optimum - The optimal solution has been found, subject to current tolerance settings.
1	Infeasible - No solution exists that satisfies all constraints.
2	Unbounded - The objective can be improved without bound.
3	Undetermined - The solution process failed.
4	Feasible - A feasible solution was found that may, or may not, be the optimal solution.
5	Infeasible or Unbounded - The preprocessor determined the model is either infeasible or unbounded. Turn off presolving and re-solve to determine which.
6	Local Optimum - Although a better solution may exist, a locally optimal solution has been found.
7	Locally Infeasible - Although feasible solutions may exist, LINGO was not able to find one.
8	Cutoff - The objective cutoff level was achieved.
9	Numeric Error - The solver stopped due to an undefined arithmetic operation in one of the constraints.

In general, if `@STATUS()` does not return a code of 0, 4, 6, or 8, then the solution is of little use and should not be trusted. In many cases Lingo will not even export data to the `@POINTER()` memory locations if `@STATUS()` does not return one of these three codes.

Troubleshooting

64-bit Lingo vs 32-bit Lingo

The Lingo API is configured to work with both 64- and 32-bit versions of Lingo. However, to use the 64-bit version of Lingo a 64-bit version of Python must be used. Similarly, the 32-bit version of Lingo requires a 32-bit version of Python. When `pip install lingo_api` runs, the version of Python associated with pip will install the appropriate bit-level version of Lingo API. To determine the version of Python associated with pip use the command: `pip -V`.

Possible errors due to misconfiguration

No Environment Variable

For 64-bit versions of Lingo the environment variable `LINGO64_19_HOME` must be set before using the Lingo API. If it is not set, you will see the error “Environment variable LINGO64_19_HOME should be set to the Lingo64_19 directory".

Similarly for 32-bit versions of Lingo the environment variable `LINGO_19_HOME` must be set before using the Lingo API. If it is not set, the error "Environment variable LINGO_19_HOME should be set to the Lingo19 directory".

Normally, Lingo’s installation program sets these environment variables, so they will not normally be of concern.

Fix Using Windows

On the command line for Windows 64:

```
> setx LINGO_19_HOME "C:\LINGO64_19"
```

On the command line for Windows 32

```
> setx LINGO64_19_HOME "C:\LINGO64_19"
```

Fix Using Linux

For administrative users:

```
$ export LINGO64_19_HOME="/opt/lingo19"
```

For standard (non-administrative) users:

```
$ export LINGO64_19_HOME="~/opt/lingo19"
```

To have this variable set automatically, add the above line to your `~/.bashrc` or `~/.bash_profile` file.

Lingo Import Error

This error will occur when the .dll (Windows), or .so (Linux) files are not where they are expected. If the .dll, or .so files are never moved or deleted this error will not occur. If, however the files have been moved then when `import lingo_api` is ran. For example, this is what the error looks like for windows 64-bit versions.

```
Lingo Import Error:
                        Make sure all the following files are present
in C:\LINGO64_19:
                         Chartdir60.dll
                         Cilkrts20.dll
                         Conopt3.dll
                         Conopt464.dll
                         Libifcoremd.dll
                         Libiomp5md.dll
                         Libmmd.dll
                         Lindo64_13_0.dll
                         Lindopr64_8.dll
                         Lingd64_19.dll
                         Lingdb64_3.dll
                         Lingf64_19.dll
                         Lingfd64_19.dll
                         Lingj64_19.dll
                         Lingoau64_14.dll
                         Lingr64_1.dll
                         Lingxl64_5.dll
                         Mosek64_9_2.dll
                         Msvcr120.dll
>>>
```

The directory `C:\LINGO64_19` is the same directory that the environment variable `LINGO64_19_HOME` points to. The .dll files are all of the files that where present in that directory when Lingo was initially installed and need to remain in that directory.

Error Codes

The Lingo API function `solve()` makes the API calls to Lingo to allocate memory, solve the model, and to deallocate the memory. These calls return an error code that is checked by `solve()`. If the error code is not 0 (no error) then Python will raise exception display the error message and end the program. The table below includes all the errors that may occur.

Value	Name	Descriptions
0	LSERR_NO_ERROR_LNG	No error.
1	LSERR_OUT_OF_MEMORY_LNG	Out of dynamic system memory.
2	LSERR_UNABLE_TO_OPEN_LOG_FILE_LNG	Unable to open the log file.
3	LSERR_INVALID_NULL_POINTER_LNG	A NULL pointer was passed to a routine that was expecting a non-NULL pointer.
4	LSERR_INVALID_INPUT_LNG	An input argument contained invalid input.
5	LSERR_INFO_NOT_AVAILABLE_LNG	A request was made for information that is not currently available.
6	LSERR_UNABLE_TO_COMPLETE_TASK_LNG	Unable to successfully complete the specified task.
7	LSERR_INVALID_LICENSE_KEY_LNG	The license key passed to *LScreateEnvLicenceLng()* was invalid.
8	LSERR_INVALID_VARIABLE_NAME_LNG	A variable name passed to *LSgetCallbackVarPrimal()* was invalid.
1000	LSERR_JNI_CALLBACK_NOT_FOUND_LNG	A valid callback function was not found in the calling Java

Here is an example of what is displayed in the terminal after a non-zero error code is returned.

```
File "C:\Users\James\Documents\GitHub\lingoapi-
python\examples\CHESS\chess.py", line 72, in <module>
    lingo.solve(model)
File "C:\Users\James\Desktop\myenv\lib\site-
packages\lingo_api\modelLoader.py", line 79, in solve
    raise LingoError(1)
lingo_api.lingoExceptions.LingoError: 1 -> Out of dynamic system
memory.
```

Type Error

The pointer dictionary `pointerDict` that is sent to the `lingo_api.Model()` needs to be a NumPy array of a numeric type, list of integers or floats, int, or float. Every element in `pointerDict` will be converted to a NumPy array of type `np.double` making it the preferred type to send. When an unsupported type is in `pointerDict` then an exception will be raised, and an error will be displayed. For example, if an element of type string is in `pointerDict` here is the error that will display:

```
Traceback (most recent call last):
  File "C:\Users\James\Documents\GitHub\lingoapi-
python\examples\CHESS\chess.py", line 72, in <module>
    lingo.solve(model)
  File "C:\Users\James\Desktop\myenv\lib\site-
packages\lingo_api\modelLoader.py", line 102, in solve
    raise TypeNotSupportedError(error)
lingo_api.lingoExceptions.TypeNotSupportedError: Pointer7 foo type:
<class 'str'> -> Unsupported type
Preferred type: NumPy Array of np.double
Other excepted: Int, floats, lists of ints or floats
```

How to Build Wheel and Install (for package managers)

To build the python package on any operating system first start by creating a whl file. From the top of the `lingoapi-python` directory run the command.

```
python -m build
```

If the command is successful, a new directory named `dist` is created in the `lingoapi-python` directory. The new directory will have two files with extension .whl and .tar.gz. For example, if you build on Windows using Python 3.10 the new directory will look like this.

```
├── dist
| ├── lingo_api-x.y.z-cp310-cp310-win_amd64.whl
| └── lingo-x.y.z.tar.gz
```

The package can now be installed locally using the command.

```
> pip install dist/*.whl
```