Python Library: Augmenting MOOS with Python

Spring 2018

Arjun Gupta, argupta@mit.edu
Department of Computer Science
MIT, Cambridge MA 02139

1 Python Library: Augmenting MOOS with Python

The goal of the `python` library is to allow users to integrate python functions and classes into their MOOSApps to make use of the wide range of available python libraries and ease of use of the python language without compromising the speed of C++.

2 Python Library Dependencies

Prior to using the `Python` Library, the following dependencies must be installed.

2.1 MAC OS X and Ubuntu

This platform requires Python2.7 (comes pre-installed on recent MacOS) as well as a number of Python packages. If you are not running on MacOS and do not already have Python2.7 installed, you can download it here:

```
https://www.python.org/downloads/release/python-2714/
```

Include the following directories in your `PYTHONPATH` environment variable:

```
{path to}/moos-ivp-argupta/src/lib_python
```

3 PythonCaller Class

The library implements a PythonCaller class which has a constructor that initializes the python interpreter, and a destructor that deallocates variables and stops the python interpreter. Once initialized, the `set_program(string program_name)` command must be called to set the python program, and then the `add_funcs(vector[string] funcs)` to initialize the functions that will be used by the PythonCaller class. Once the program and functions are loaded, the functions can be called through the `run(string method, vector[string] sArgs, vector[PyObject*] pArgs, vector[vector[string]] vArgs)` command. Where *sArgs*, *pArgs*, and *vArgs*, are the arguments passed to the function *method*. The arguments are passed to the function with *pArgs* first, *sArgs* second, and *vArgs* last. Where *pArgs* are meant to hold arguments that are python objects, *sArgs* are meant to hold arguments that are string objects (can be converted to intergers in python), and *vArgs* is meant to hold vector inputs (python lists) to the function. The *run* function returns a *PyObject*∗ which can then be converted to a C++ style variable for use in the rest of the C++ code.

4 Functions

Constructor Creates the PythonCaller object and begins the Python interpreter

setProgram Takes in a string that is the name of the program (without the .py suffix) and loads the python file specified

addFuncs Takes a list of strings that are the names of possible functions to be executed by the PythonCaller. Loads the python functions to be called later

run takes in the function name as a string and then different lists of arguments. The first set of arguments is a vector or strings, the second is a vector of PyObject *, and the last is a vector of a vector of strings for Python list arguments. The run function passes the PyObject * parameters in order, then the String parameters in order, then the vector parameters in order as arguments to the specified function. It calls the function with these arguments and then returns a PyObject * holding the result.

Deconstructor deallocates the local PyObject * variables and then stops the Python interpreter.