mtypes Documentation

Release 0.1.1

Quansight-Labs

Apr 27, 2020

Contents:

CHAPTER 1

mtypes API Specification

mtypes.mtype is a metaclass that allows you to associate extra C-level information with your type objects like classes within Python. Also, objects instantiated from those types also have C-level data associated with them.

1.1 PyMType_Type

```
PyMTypeObject PyMType_Type;
```

PyMType_Type is the base class for everything. This will be used as the main type to generate all other mtypes.

1.2 PyMTypeObject

```
typedef struct _mtypeobject
{
    PyHeapTypeObject ht_obj;
    // The functions used for marshaling this type in and out of Python.
    boxfunction box;
    unboxfunction unbox;
    PyMTypeFunction *mt_funcs;
    void *mt_data;
} PyMTypeObject;
```

PyMTypeObject is essentially a PyTypeObject that allows for extra functionality, effectively:

- Storing extra data within the object within the mt_data member.
- Marshalling to and from C with the __cdict__ protocol via the box and unbox methods.

1.3 PyMTypeFunction

```
typedef struct _mfunc
{
    // Analogous to ht_name, ht_slots and ht_qualname in PyHeapTypeObject
    char *mt_name;
    mt_func mt_slot;
    char *mt_qualname;
    PyMTypeArgument *arguments;
    PyMTypeObject *mt_rettype;
} PyMTypeFunction;
```

`PyMTypeFunction` Is a struct that simply contains the functions that are implemented into the mtype. This will have all the methods that relate to that specific function, and will associate the functions with its basic attributes.

- `mt_name` is the argument that stores the name of the mtype.
- `mt_slots` is of the type mt_func, defined in `_mtypes.h`. It will store the function pointer (and point to the C-level function).
- `mt_qualname` is used to get information about the function
- `PyMTypeArgument` is explained below. It contains arguments that define the type.
- `mt_rettype` is the return type of the function.

1.4 PyMTypeArgument

```
typedef struct _margument
{
    // These fields are used to function signatures for a given function.
    char *name;
    PyMTypeObject *type;
} PyMTypeArgument;
```

`PyMTypeArgument` simply contains the function signature, unique to each function. - `char *name` is the name of the function itself.

This is a Non-NULL value, returning an empty sting.

- `PyMTypeObject *type` is a pointer to the type of the argument.

1.5 PyMObject

```
typedef struct _mobject
{
    PyObject obj;
    void *m_data;
} PyMObject;
```

`PyMObject` It acts as an intermediary between the Python level object and the C level object. It contains:

- `PyObject`, which is the Python-Level Object
- `void *m_data` which stores the C-level data that is needed for this object to be represented.

We will use the `box` and `unbox` methods to interface between the Pythonic and C level objects via the `__cdict__` protocol.

1.6 `box` and `unbox`

These functions will be used as the layer of translation between C structs and Python objects.

```
typedef PyObject *(*boxfunction)(PyMTypeObject *type, void *data);
typedef int (*unboxfunction)(PyObject *obj, void *data);
```

1.6.1 `boxfunction`

The box function is defined as the layer that converts a C struct into a Python object. The function will perform error checking and will return an *instance* of a `PyMType`.

Input Arguments

- `PyMTypeObject *type` : The type that the C struct should be marshalled into.
- `void *data`: A pointer to the data that needs to be marshalled and converted to a Python Object.

Output Argument

- `PyObject *out`: A pointer to a `PyObject` initalized on the heap from the C struct. `NULL` indicates failure. If returning `NULL`, a Python exception must be set. The returned object must be a pointer to either a `PyMTypeObject` or `PyMObject` which has the Python type of `type`.

1.6.2 `unboxfunction`

The `unbox` function is a C function that takes in the Python object to be marshalled into a C struct.

Input Arguments

- `PyObject *obj`: The object to be marshalled into C. Its type must be an instance of `PyMTypeObject`.
- `void *data`: A pointer to the C struct to put the data into.

Output Argument

- `int return_code`: Must be `-1` on failure and `0` on success. If returning `-1`, a Python exception must be set.

1.7 `__cdict__` protocol

The `__cdict__` protocol will be used as an intermediary between Python level objects and C structs. This will allow a specific type signature to be passed from the Python Object *into* the `unbox` function, then passing to a `ctypes` method, and then calling the `box` function eventually returning a `PyMObject`.

This dict provides the `boxfunction` and the `unboxfunction` with the proper associated C type signature.

```
from typing import Tuple, Dict
from ctypes import CFUNCTYPE
from mtypes import mtype

class MarshalledClass(metaclass=mtype):
    __cdict__:
        Dict[str, # Lookup by name
            Dict[
                paramflags_type, # Lookup by signature
                CFUNCTYPE,       # Implementation
            ],
        ]
```

CHAPTER 2

Indices and tables

- genindex
- modindex
- search