Article
Eduard Lebedyuk · Jul 4, 2022 3m read

Calling classmethods with Native API for Python

InterSystems Native SDK for Python is a lightweight interface to InterSystems IRIS APIs that were once available only through ObjectScript.

I'm especially interested in the ability to call ObjectScript methods, class methods, to be precise. It works, and it works great, but by default, calls only support scalar arguments: strings, booleans, integers, and floats.

But if you want to:
- Pass or return structures, such as dicts or lists
- Pass or return streams

You'll need to write some glue code or take this project (installs with pip install edpy). edpy package gives you one simple signature:

```
call(iris, class_name, method_name, args)
```

which allows you to call any ObjectScript method and get results back.

Import it like this:

```
from edpy import iris
```

call accepts 4 required arguments:
- iris - a reference to an established IRIS object
- classname - IRIS class to call
- methodname - IRIS method to call
- args - list of 0 or more arguments

Arguments

Each argument can be one of:

- string (any length, if larger than $$$MaxStringLength or 3641144 symbols, it would be automatically converted into a stream
- boolean
- integer
- float
- dict (converts into a dynamic object)
- list or tuple (converts into a dynamic array)

dict, list, and tuple arguments can recursively contain other dicts, lists, and tuples (while memory lasts).

Return value

In return, we expect either a dynamic object/array or a JSON string/stream. In that case, edpy would first convert it into a Python string and, if possible, interpret it as a Python dict or list. Otherwise, the result would be returned to the caller as is.

That's about it, but let me give you some examples of ObjectScript methods and how to call them using this Python function.

Example 1: Pong

```
ClassMethod Test(arg1, arg2, arg3) As %DynamicArray
{
    return [(arg1), (arg2), (arg3)]
}
```

Call with:

```
>>> iris.call(iris_native, "User.Py", "Test", [1, 1.2, "ABC"])
[1, 1.2, 'ABC']
```

No surprise here. Arguments are packed back into an array and returned to the caller.

Example 2: Properties

```
ClassMethod Test2(arg As %DynamicObject) As %String
{
    return arg.Prop
}
```

Call like this:

```
>>> iris.call(iris_native, "User.Py", "Test2", [{"Prop":123}])
123
```

Now for a more embedded invocation:

```
>>> iris.call(iris_native, "User.Py", "Test2", [{"Prop":{"Prop2":123}}])
{'Prop2': 123}
```

If a property is too long that's also okay - streams will be used to send it to IRIS and/or back:

```
ret = iris.call(iris_native, "User.Py", "Test2", [{"Prop":"A" * 10000000}])
>>> len(ret)
10000000
```

If you need guaranteed streams on the InterSystems IRIS side, you can use %Get:

```
set stream = arg.%Get("Prop",,"stream")
```

If the stream is base64 encoded you can automatically decode it with:

```
set stream = arg.%Get("Prop",,"stream<base64")
```

Example 3: String or Stream

```
ClassMethod Test3(arg As %Stream.Object) As %String
{
    set file = ##class(%Stream.FileCharacter).%New()
    set file.TranslateTable = "UTF8"
    set filename = ##class(%File).ManagerDirectory() _ "test.txt"
    do file.LinkToFile(filename)
    if $isObject(arg) {
        set sc = file.CopyFromAndSave(arg)
    } else {
        do file.Write(arg)
        set sc = file.%Save()
    }
    if $$$ISERR(sc) {
        set jsonret = {"status":0, "payload":($system.Status.GetErrorText(sc))}
    } else {
        set jsonret = {"status":1}
    }
    quit jsonret.%ToJSON()
}
```

Here we write either a string or a stream to <mgr>test.txt.

```
>>> iris.call(iris_native, "User.Py", "Test3", ["&#x1f60a;"])
{'status': 1}
```

Note: in all code samples "&# x1f642;" is entered as .

And if I open the file I'll see one and not two ?? - so we preserve encoding.

```
>>> iris.call(iris_native, "User.Py", "Test3", ["&#x1f642;" * 10000000])
{'status': 1}
```

I'll omit the file output for brevity, but it's there.

Finally, by passing a dynamic object or array inside, you can avoid string/stream dichotomy altogether, even if you don't know if the property would be shorter or longer than the string limit. In that case, you can always get your suspect property like a stream.

Example 4: Return streams

```
ClassMethod Test4(arg As %DynamicArray) As %String
```

```
{
    return arg.%Get(0)
}
```

Here's how that looks like:

```
>>> ret = iris.call(iris_native, "User.Py", "Test4", [["&#x1f60a;" * 10000000]])
>>> len(ret)
10000000
>>> ret[:5]
'&#x1f60a;&#x1f60a;&#x1f60a;&#x1f60a;&#x1f60a;'
```

One more thing

There's also a getiris(ip="localhost", port=1972, namespace="USER", username="SYSTEM", password="SYS") function which gets you a working IRIS object.
So here's a complete example, if you want to try it yourself:

First load the User.Py class and install edpy python library:

```
pip install edpy
```

And then in python call:

```
from edpy import iris
iris_native = iris.get_iris()
iris.call(iris_native, "User.Py", "Test",  [1, 1.2, "ABC"])
iris.call(iris_native, "User.Py", "Test2", [{"Prop":123}])
iris.call(iris_native, "User.Py", "Test2", [{"Prop":{"Prop2":123}}])
ret2 = iris.call(iris_native, "User.Py", "Test2", [{"Prop":"A" * 10000000}])
iris.call(iris_native, "User.Py", "Test3", ["&#x1f60a;"])
iris.call(iris_native, "User.Py", "Test3", ["&#x1f60a;" * 10000000])
ret4 = iris.call(iris_native, "User.Py", "Test4", [["&#x1f60a;" * 10000000]])
```

Conclusion

Native SDK for Python is a powerful tool, providing you a complete and unrestricted access to InterSystems IRIS. I hope this project can save you some time marshaling InterSystems IRIS calls. Is there some combination of method arguments it does not support? If so, share in the comments how you call them.

Links

- Native SDK Docs
- User.Py class
- Repo

#Python #InterSystems IRIS