Article
Nikita Mullin · Jun 1, 2020 4m read

Open Exchange

# How I added ObjectScript to Jupyter Notebooks

Jupyter Notebook is an interactive environment consisting of cells that allow executing code in a great number of different markup and programming languages.

To do this Jupyter has to connect to an appropriate kernel. There was no ObjectScript Kernel, that is why I decided to create one.

You can try it out here.

Here's a sneak peek of the results:

## Jupyter Kernels 101

There are several ways to create a Jupyter Kernel. I decided to make a Python wrapper kernel.

We have to create a subclass of ipykernel.kernelbase.Kernel and implement the doexecute method which receives a code to be executed in a particular language.

So, the general idea is to get a piece of ObjectScript code, somehow execute it and return the results to our notebook.

But how do we that exactly? Let's try and break that down even further.

## Sending ObjectScript code to IRIS

To begin with, we have to send our piece of code to IRIS. This is where IRIS Native API for Python comes in.

All we have to do is import irisnative package, then establish a connection:

```
def get_iris_object():
  # Create connection to InterSystems IRIS
  connection = irisnative.createConnection('iris', 51773, 'IRISAPP', '_SYSTEM', 'SYS'
)

  # Create an iris object
  return irisnative.createIris(connection)
```

After that, we can use the connection to call classes that are stored in the IRIS database.

```
def execute_code(self, code):
        class_name = "JupyterKernel.CodeExecutor"
        return self.iris.classMethodValue(class_name, "CodeResult", code)
```

What are these CodeExecutor class and CodeResult method used for?

Let's take a look.

## Excecuting ObjectScript code

The purpose of this class is to execute a line of ObjectScript code and return a JSON object with the results of execution. We pass our code to CodeResult in a variable vstrCommand.

We start with redirecting IO to the current routine, after that we execute passed code via XECUTE command, redirect IO back to the original and then return the results.

```
Include %sySystem

Class JupyterKernel.CodeExecutor
{

ClassMethod CodeResult(vstrCommand As %String) As %String [ ProcedureBlock = 0 ]
{
        set tOldIORedirected = ##class(%Device).ReDirectIO()
        set tOldMnemonic = ##class(%Device).GetMnemonicRoutine()
        set tOldIO = $io
        try {
            set str=""
            set status = 1
            //Redirect IO to the current routine - makes use of the labels defined be
low
            use $io::("^"_$ZNAME)

            //Enable redirection
            do ##class(%Device).ReDirectIO(1)

            XECUTE (vstrCommand)

        } catch ex {
            set str = ex.DisplayString()
            set status = 0
        }

        //Return to original redirection/mnemonic routine settings
        if (tOldMnemonic '= "") {
            use tOldIO::("^"_tOldMnemonic)
        } else {
            use tOldIO
        }
        do ##class(%Device).ReDirectIO(tOldIORedirected)

        quit {"status":(status), "out":(str)}.%ToJSON()

rchr(c)
    quit
rstr(sz,to)
    quit
wchr(s)
    do output($char(s))
    quit
wff()
```

```
    do output($char(12))
    quit
wnl()
    do output($char(13,10))
    quit
wstr(s)
    do output(s)
    quit
wtab(s)
    do output($char(9))
    quit
output(s)
    set str = str _ s
    quit
}

}
```

## Displaying the results

So, we've executed a piece of ObjectScript code, now what? Well, we have to display the results.

If there were no exceptions, we just display the results line by line.

However, if our a passed piece of code did raise an exception, we stop the execution, display the failed line's number, itself, and the raised exception.

## Launching the app

You can try this kernel yourself and here's how.

## Prerequisites

Make sure you have git and Docker installed.

Clone/git pull the repo into any local directory e.g. like it is shown below:

```
$ git clone https://github.com/Vekkby/objectsriptkernel.git
```

Open the terminal in this directory and run:

```
$ docker-compose up -d --build
```

## How to Work With it

You may access the notebook server from the browser using

```
localhost:8888
```

There's a sample notebook named 'hello.ipynb' in the 'work' directory.

## Vote

This app is a part of IRIS Native API contest.
You can vote for this app here.

#API #Python #InterSystems IRIS
Check the related application on InterSystems Open Exchange

**Source URL:**https://community.intersystems.com/post/how-i-added-objectscript-jupyter-notebooks