Article
Evgeny Shvarov · Oct 24, 2022 4m read

Open Exchange

Embedded Python Template

Hi developers!

Let me share with you a minimal embedded python template, that I can recommend as a starting point for any general project with InterSystems IRIS that will use embedded python.

Features:

- Embedded Python ready;
- Examples of 3 ways of Embedded python development;
- VSCode development ready;
- Docker enabled;
- Online demo enabled;
- ZPM First development ready.

Let's discuss the features below!

Let's first talk about Embedded Python. This feature comes with InterSystems IRIS 2021.2 and allows to develop solutions using InterSystems IRIS with python. IRIS 2021.2 and above allows python scripts execution in a shared memory environment with InterSystems IRIS which gives unique options to python developers in using the database where code is close to data.

3 modes of developing with Embedded python

Calling python libs from ObjectScript

This becomes possible because of %SYS.Python class, which allows to import python libs and call python via ObjectScirpt. Documentation, Example. See the code below:

```
ClassMethod Today() As %Status
{
  Set sc = $$$OK
  Set dt = ##class(%SYS.Python).Import("datetime")
  write dt.date.today().isoformat()
  Return sc
}
```

Writing ObjectScript class methods in python

Indeed, now developers can put a [Language=python] tag in the method signature and code in pure python. There is also a helper python lib "iris" that can be used to refer to ObjectScript classes and globals. Documentation, Example, Sample code:

```
ClassMethod CreateRecordPython(propValue As %VarString, ByRef id As %Integer) [ Langu
```

```
age = python ]
{
    import iris
    obj=iris.cls(__name__)._New()
    obj.Test=propValue
    sc=obj._Save()
    id=obj._Id()
    return sc
}
```

Coding InterSystems IRIS solutions in pure python

This is the third option of how developers can deal with IRIS. Here python script needs to be connected to IRIS, and this can be done via ENV variables and CallIn service "On", see the details below. Once set up python script is executed in the shared memory with IRIS. Here "iris" lib becomes very helpful as well. Documentation, Example.

```
def create_rec(var):
    obj=iris.cls('dc.python.PersistentClass')._New()
    obj.Test=var
    obj._Save()
    id=obj._Id()
    return id

# test record creation
from datetime import datetime
now=str(datetime.now())
print("Creating new record in dc.python.PersistentClass")
print(create_rec(now))

## run SQL and print data
def run_sql(query):
    rs=iris.sql.exec(query)
    for idx, row in enumerate(rs):
        print(f"[{idx}]: {row}")

query="Select * from dc_python.PersistentClass"
print("Running SQL query "+query)
run_sql(query)
```

Docker enabled

The template repository runs IRIS in a container and sets up all the necessary for Embedded Python tweaks.

Environment variables. Embedded Python needs certain Env variables set up to connect to IRIS and run python scripts. Here is the settings that help with it in dockerfile:

```
# init Python env
ENV PYTHON_PATH=/usr/irissys/bin/irispython
ENV SRC_PATH=/irisrun/repo
ENV IRISUSERNAME "SuperUser"
ENV IRISPASSWORD "SYS"
ENV IRISNAMESPACE "USER"
```

Also, Embedded python needs CallIn service "on", this happens in the iris.script during the docker build phase:

```
; enabling callin for Embedded Python
    do ##class(Security.Services).Get("%Service_CallIn",.prop)
    set prop("Enabled")=1
    set prop("AutheEnabled")=48
    do ##class(Security.Services).Modify("%Service_CallIn",.prop)
```

Also, your solution may need some python libs to be installed. This is served via requirements.txt in the root of the repo and a pip3 call in the dockerfile:

```
pip3 install -r requirements.txt && \
```

VSCode development ready

It is very convenient to develop in VSCode using docker. If you want to develop in docker IRIS solution with Embedded python VSCode needs to be switched into Devcontainer mode. To make it happen introduce the devcontainer.json file into .devcontainer folder. It describes which docker service it needs to work with (iris in our case) and this helps to run python scripts in VSCode that will be served by the Python engine used by IRIS that is running in the container. devcontainer.json file also has a section with a description of which extensions need to be used in container mode:

```
"extensions": [
        "ms-python.python",
        "ms-python.vscode-pylance",
        "intersystems-community.vscode-objectscript",
        "intersystems.language-server",
        "intersystems-community.servermanager",
        "ms-vscode.docker"
    ],
```

Installing Embedded Python solutinos via ZPM

This template is set up as a "ZPM first" development repository. This means that all the developed code is already described in module.xml and being installed as ZPM module every time docker image is being built meaning everytime developer starts coding with the following line in iris.script:

```
zpm "load /home/irisowner/irisbuild/ -v":1:1
```

And Embedded python code is described in the ZPM module as well - it is being installed via FILECOPY:

```
<FileCopy Name="python/" Target="${libdir}python/"/>
```

This expression says that we want to package all the python scripts under /python folder in the repo and install it in python/ folder in a libdir of the target IRIS installation. If python scripts are copied under ${libdir}python/ folder they become available for import call either from ObjectScirpt or Python in the target IRIS machine.

NB. Make sure you follow name uniqueness with your folder names for python scripts to not override other python code accidentally.

I hope the template will be useful for you. Feedback and especially pull requests a very welcome!

#Development Environment #Embedded Python #InterSystems IRIS #VSCode
Check the related application on InterSystems Open Exchange