Article
Eduard Lebedyuk · Oct 10, 2019 7m read

Open Exchange

Python Gateway III: Basic functionality

This series of articles would cover Python Gateway for InterSystems Data Platforms. Execute Python code and more from InterSystems IRIS. This project brings you the power of Python right into your InterSystems IRIS environment:

- Execute arbitrary Python code
- Seamlessly transfer data from InterSystems IRIS into Python
- Build intelligent Interoperability business processes with Python Interoperability Adapter
- Save, examine, modify and restore Python context from InterSystems IRIS

Other articles

The plan for the series so far (subject to change).

- Part I: Overview, Landscape and Introduction
- Part II: Installation and Troubleshooting
- Part III: Basic functionality <-- you're here
- Part IV: Interoperability Adapter
- Part V: Execute function
- Part VI: Dynamic Gateway
- Part VII: Proxy Gateway
- Part VIII: Use cases and ML Toolkit

Intro

You now have Python gateway installed and validated that it works, time to start using it!
In this article I would cover the main interface to Python, which is isc.py.Main. It offers these methods (all return %Status), which can be separated into

- Code execution
- Data Transfer
- Auxiliary

Code execution

These methods allow execution of arbitrary Python code.

SimpleString

SimpleString is the main method. It accepts 4 optional arguments:
- code - string of code to execute. Separate several lines with $c(10).
- returnVariable - name of a variable to return.
- serialization - how to serialize returnVariable. 0 - string serialization (default), 1 - repr serialization.
- result - pass variable by reference. returnVariable is written into it.

Here's an example:

```
set sc = ##class(isc.py.Main).SimpleString("x='HELLO'", "x", , .var)
```

In this example we assign Python variable x value Hello and we want to return Python x variable into ObjectScript var variable.

ExecuteCode

ExecuteCode is a safer alternative to SimpleString without SimpleString limitations.
Strings are limited by 3 641 144 characters, and if you want to execute a longer piece of code you need to use streams.
It accepts two arguments:
- code - string or stream of Python code to execute.
- variable - (optional) set result of code execution into this Python variable.

Here's how to use it:

```
set sc = ##class(isc.py.Main).ExecuteCode("2*3","y")
```

In this example we multiply 2 by 3 and write the result into Python y variable.

Data Transfer

Transfer data into and from Python.

Python -> InterSystems IRIS

There are 4 ways to get variable value from Python to InterSystems IRIS, depending on the serialization you need:
- String for simple datatypes and debugging.
- Repr for storage of simple objects and debugging.
- JSON for easy maipulation of data on InterSystems IRIS side.
- Pickle for persisting objects.

These methods allow getting variables from Python as string or streams.

- GetVariable(variable, serialization, .stream, useString) - get serialization of variable in stream. If useString is 1 and variable serialization can fit into string then string is returned instead of the stream.
- GetVariableJson(variable, .stream, useString) - get JSON serialization of variable.
- GetVariablePickle(variable, .stream, useString, useDill) - get Pickle (or Dill) serialization of variable.

Let's try to get our y variable.

```
set sc = ##class(isc.py.Main).GetVariable("y", , .val, 1)
w val
>6
```

InterSystems IRIS -> Python

And finally let's load some data into Python from InterSystems IRIS.

- ExecuteQuery(query, variable, type, namespace) - create resultset (pandas dataframe or list) from sql query and set it into variable. isc.py package must be available in namespace.
- ExecuteGlobal(global, variable, type, start, end, mask, labels, namespace) - transfer global data (from start to end) to Python variable of type: list of tuples or pandas dataframe. For mask and labels arguments specification check class docs and Data Transfer docs.
- ExecuteClass(class, variable, type, start, end, properties, namespace) - transfer class data to Python list of tuples or pandas dataframe. properties - comma-separated list of properties to form dataframe from. * and ? wildcards are supported. Defaults to * (all properties). %%CLASSNAME property is ignored. Only stored properties can be used.
- ExecuteTable(table, variable, type, start, end, properties, namespace) - transfer table data to Python list of tuples or pandas dataframe.

ExecuteQuery is universal (any valid SQL query would be transfered into Python). ExecuteGlobal and its wrappers ExecuteClass and ExecuteTable, however, operate with a number of limitations. But they are much faster (3-5 times faster than ODBC driver and 20 times faster than ExecuteQuery). More information in Data Transfer docs. All these methods support data transfer from any local namespace. isc.py package must be available in namespace.

ExecuteQuery

ExecuteQuery(query, variable, type, namespace) - transfer results from any valid SQL query into Python. It is the slowest method of data transfer. Use it if ExecuteGlobal and its wrappers are unavailable.

Arguments:
- query - sql query
- variable - target variable on a Python side
- type - list or Pandas dataframe

ExecuteGlobal

ExecuteGlobal(global, variable, type, start, end, mask, labels, namespace) - transfer global data to Python.

Arguments:

- global - global name without ^
- variable - target variable on a Python side
- type - list or Pandas dataframe
- start - initial global key. Must be integer.
- end - final global key. Must be integer.
- mask - string, mask for global values. Mask may be shorter than the number of global value fields (in this case fields at the end would be skipped). How to format mask:
    - + use field as is
    - - skip field
    - b - boolean (0 - False, anything else - True)
    - d - date (from $horolog, on Windows only from 1970, on Linux from 1900 see notes for details)
    - t - time ($horolog, seconds since midnight)
    - m - (moment) timestamp string in YEAR-MONTH-DAY HOUR:MINUTE:SECOND format.
- labels - %List of column names, first element is key column name. Therefore: List length must be mask symbol length + 1.

ExecuteClass

Wrapper for ExecuteGlobal. Effectively it parses compiled class definition, constructs ExecuteGlobal arguments and calls it.

ExecuteClass(class, variable, type, start, end, properties, namespace) - transfer class data to Python list of tuples or pandas dataframe. properties - comma-separated list of properties to form dataframe from. * and ? wildcards are supported. Defaults to * (all properties). %%CLASSNAME property is ignored. Only stored properties can be used.

Arguments:
- class - class name
- variable - target variable on a Python side
- type - list or Pandas dataframe
- start - initial object id. Must be integer.
- end - final object id. Must be integer.
- properties - comma-separated list of properties to form dataframe from. * and ? wildcards are supported. Defaults to * (all properties). %%CLASSNAME property is ignored. Only stored properties can be used.

All properties transferred as is except properties of %Date, %Time, %Boolean and %TimeStamp types. They are converted to respective Python datatypes.

ExecuteTable

Wrapper for ExecuteClass. Translates table name to class name and calls ExecuteClass. Signature:

ExecuteTable(table, variable, type, start, end, properties, namespace) - transfer table data to Python list of tuples or pandas dataframe.

Arguments:
- table - table name.

Other arguments are passed as is to ExecuteClass.

Notes

- ExecuteGlobal, ExecuteClass and ExecuteTable generally offer the same speed (as the time to parse class definition is negligible).
- ExecuteGlobal is up to 20 times faster than ExecuteQuery on measurable workloads (>0.01 second).
- ExecuteGlobal, ExecuteClass and ExecuteTable only work on the globals with this structure: ^global(key) = $lb(prop1, prop2, ..., propN) where key must be an integer.
- For ExecuteGlobal, ExecuteClass and ExecuteTable supported %Date range equals mktime range (windows: 1970-01-01, linux 1900-01-01, mac). Use %TimeStamp to transfer dates outside of this range. Or use pandas dataframe (as this is a list limitation).
- For ExecuteGlobal, ExecuteClass and ExecuteTable all arguments besides the source (global, class, table) and variable are optional.

Examples

Let's say we have isc.py.test.Person class. Here's how we can use all methods of data transfer:

```
// All the ways to transfer data
set global = "isc.py.test.PersonD"
set class = "isc.py.test.Person"
set table = "isc_py_test.Person"
set query = "SELECT * FROM isc_py_test.Person"

// Common arguments
set variable = "df"
set type = "dataframe"
set start = 1
set end = $g(^isc.py.test.PersonD, start)

// Approach 0: ExecuteGlobal without arguments
set sc = ##class(isc.py.Main).ExecuteGlobal(global, variable _ 0, type)

// Approach 1: ExecuteGlobal with arguments
// For global transfer labels are not calculated automatically
// globalKey - is global subscript
set labels = $lb("globalKey", "Name", "DOB", "TS", "RandomTime", "AgeYears", "AgeDeci
mal", "AgeDouble", "Bool")

// mask is 1 element shorter than labels because "globalKey" is global subscript labe
l
// Here we want to skip %%CLASSNAME field
set mask = "-+dmt+++b"

set sc = ##class(isc.py.Main).ExecuteGlobal(global, variable _ 1, type, start, end, m
ask, labels)

// Approach 2: ExecuteClass
set sc = ##class(isc.py.Main).ExecuteClass(class, variable _ 2, type, start, end)

// Approach 3: ExecuteTable
set sc = ##class(isc.py.Main).ExecuteTable(table, variable _ 3, type, start, end)

// Approach 4: ExecuteTable
set sc = ##class(isc.py.Main).ExecuteQuery(query, variable _ 4, type)
```

You can call this method: do ##class(isc.py.test.Person).Test() to check how these data transfer methods work.

Auxiliary

Support methods.

- GetVariableInfo(variable, serialization, .defined, .type, .length) - get info about variable: is it defined, type and serialized length.
- GetVariableDefined(variable, .defined) - is variable defined.
- GetVariableType(variable, .type) - get variable FQCN.
- GetStatus() - returns last occurred exception in Python and clears it.
- GetModuleInfo(module, .imported, .alias) - get module alias and is it currently imported.
- GetFunctionInfo(function, .defined, .type, .docs, .signature, .arguments) - get function information.

Summary

Python Gateway allows seamless integration between InterSsytems IRIS and Python. Use it to execute code and transfer data bidirectionally.

Links

- Python Gateway
- Install Python 3.6.7 64 bit
- Python documentation/tutorial

Illustrated guide

There's also illustrated guide in ML Toolkit user group. ML Toolkit user group is a private GitHub repository set up as part of InterSystems corporate GitHub organization. It is addressed to the external users that are installing, learning or are already using ML Toolkit components including Python Gateway. To join ML Toolkit user group, please send a short e-mail at the following address: MLToolkit@intersystems.com and indicate in your e-mail the following details (needed for the group members to get to know and identify you during discussions):

- GitHub username
- Full Name (your first name followed by your last name in Latin script)
- Organization (you are working for, or you study at, or your home office)
- Position (your actual position in your organization, or “ Student ” , or “ Independent ”)
- Country (you are based in)

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