InterSystems™
IRIS Data Platform

Using Python Gateway Connections

Version 2020.3
2021-02-04

InterSystems Corporation   1 Memorial Drive   Cambridge MA 02142   www.intersystems.com

Using Python Gateway Connections
InterSystems IRIS Data Platform   Version 2020.3   2021-02-04
Copyright © 2021 InterSystems Corporation
All rights reserved.

Table of Contents

List of Figures

About This Book

InterSystems IRIS® *Python gateway connections* allow you to create an ObjectScript *proxy object* linked to a corresponding Python object. The proxy object has the same set of methods and properties as the Python object, and is functionally identical to the original. Any call to the proxy object is echoed by the Python object, and returned values are immediately sent back to the proxy.

The proxy and the Python object communicate over TCP/IP using the InterSystems *gateway connection* communications protocol, enabled on the Python side by an *External Python Server*. The gateway connection provides a completely transparent bidirectional link between the two objects even when they are running on two different machines.

The following chapters describe how to use Python gateway connections:

- Introduction to Gateway Connections provides an overview of gateway connections, proxy objects, and External Python Servers.
- Connecting and Using Proxy Objects — demonstrates how to establish gateway connections and use ObjectScript proxy objects to control Python objects.
- Managing External Python Servers explains the Management Portal pages that allow you to define External Python Server parameters and control gateway connections.

There is also a detailed Table of Contents.

1 Introduction to Gateway Connections

InterSystems IRIS® provides a way for you to create an object in a Python process and control it from a corresponding *proxy object* in an ObjectScript process. The Python object and the ObjectScript proxy exchange messages over TCP/IP using the InterSystems *gateway connection* communications protocol, which is enabled on the Python side by the InterSystems *External Python Server*. The following sections describe how gateway connections work.

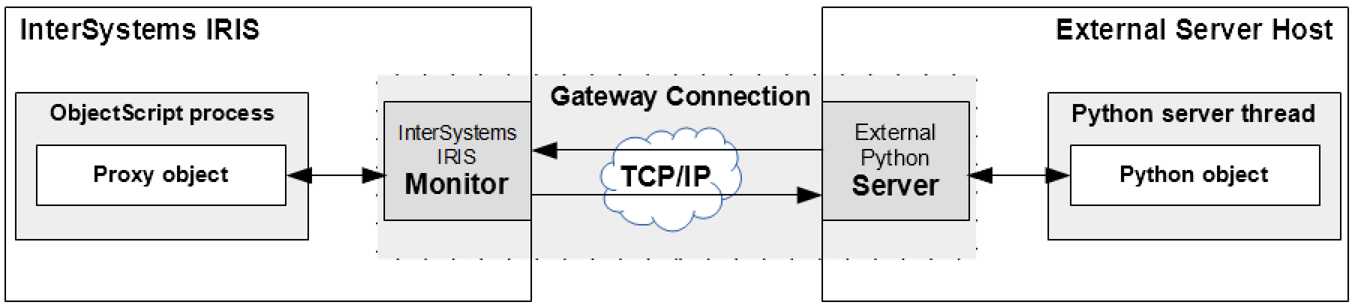
- Gateway Connection Overview
- Using Proxy Objects
- External Python Server Installation

1.1 Gateway Connection Overview

In practice, you can think of a gateway connection as a simple black box that allows a Python object and an ObjectScript proxy to exchange messages. Each gateway connection consists of the following components:

- The *Gateway Monitor* process runs in an InterSystems IRIS namespace, and connects to one or more ObjectScript proxy objects in other processes.
- The *External Python Server* is a Python process that runs on the host machine and connects to one or more Python objects. Each Python object runs in a separate thread spawned by the external server process.
- A *TCP/IP connection* allows monitor and server to exchange messages using the InterSystems gateway connection protocol.

Figure 1–1: Gateway connection between InterSystems IRIS and an external Python server



The External Python Server is a Python process running on the host machine. At startup, the external server accepts an argument for a port number, and listens on that port for a connection request. To establish a gateway connection, InterSystems IRIS starts a local monitor process that uses the same port, and sends the request that connects it to the external server.

The gateway connection provides a completely transparent bidirectional link between Python and ObjectScript processes whether they are running on the same machine or two different machines. The connection is reentrant, allowing host applications to manipulate ObjectScript objects and ObjectScript applications to manipulate host objects, with all processes using the same connection context (database, session and transaction) and physical connection.

1.2 Using Proxy Objects

Once a gateway connection is established, a single ObjectScript method call generates both a Python object and the linked ObjectScript proxy object. The ObjectScript proxy is created by introspecting the specified Python class and generating an identical set of methods and properties. At the same time on the host machine, an instance of the Python class is created and linked to the proxy. Any call to the proxy object is immediately echoed by the Python object, and any values returned by the Python object are immediately available to the proxy.

In the following example, the PersonDemo.py module contains class Person with properties *name* and *age*, and method *displayProperties()*. The ObjectScript proxy has identical methods and properties, and can be treated just as if it were an instance of the original Python class:

Creating and using an ObjectScript proxy object

```
// Create a proxy for a Person object
  set person = ##class(%Net.Remote.Object).%New(gateway,"PersonDemo.Person")

// Use properties name and age, and method displayProperties().
  set person.name = "George"
  set person.age = 99
  write !,"Name and age: ",person.displayProperties()
```

Prints:

```
  Name and age: George, Age: 99
```

See “Connecting and Using Proxy Objects” for more details and examples.

1.3 External Python Server Installation

The External Python Server requires Python 3.6.6 or higher, but does not have to be installed on the same machine as your instance of InterSystems IRIS. The installation file is intersystems_irispython-3.2.0-py3-none-any.whl, located in <install-dir>\dev\python\ (where <install-dir> is the root directory of your InterSystems IRIS instance).

Install the External Python Server package with the following command:

```
python -m pip install --upgrade <path>\intersystems_irispython-3.2.0-py3-none-any.whl
```

Do not use the `--user` option.

Tip: Display your InterSystems IRIS root path by entering the following command at the InterSystems Terminal:

```
write $SYSTEM.Util.InstallDirectory()
```

1.3.1 InterSystems IRIS Requirements

To communicate with an External Python Server, your instance of InterSystems IRIS must meet the following requirements:

- InterSystems IRIS version must be 2020.3 or higher.
- At least one named set of External Python Server configuration settings must be defined before a gateway connection can be established.

See "Managing External Python Servers" for instructions on how to specify configuration settings and enable a gateway connection.

2

Connecting and Using Proxy Objects

This chapter demonstrates how to establish a gateway connection to an External Python Server instance, and how to create and use ObjectScript proxy objects.

- Creating a Gateway Connection — demonstrates how to start a gateway connection
- Specifying Class Paths — how to specify paths for Python classes.
- Creating and Using Proxy Objects — describes how to create a linked pair of proxy and Python objects for a specified class, and demonstrates how to use the proxy.
- The demo Package — lists the tiny Python package used in these examples.

2.1 Creating a Gateway Connection

To create the connection:

- Create a Gateway instance. A gateway connection (including named definition) is encapsulated in an instance of %Net.Remote.Gateway.
- Get named server info. Host and port values for PyGate are returned by a call to %Net.Remote.Service method **OpenGateway()**.
- Call Gateway **%Connect()**. Creates the gateway connection to external server on specified port

The following example assumes that an External Python Server process named PyGate is running (see the chapter on "Managing External Python Servers").

Read the named External Python Gateway definition and create the gateway connection

```
# get settings for "PyGate" external server
  set status = ##class(%Net.Remote.Service).OpenGateway("PyGate",.PyServer)
  set host = PyServer.Server
  set port = PyServer.Port

# create a gateway connection to the PyGate External Python Server
  set status = gateway.%Connect(host, port)
```

2.2 Specifying Class Paths

You must specify a path for each module that contains a class you want to import. In this example, the fully qualified path to PersonDemo.py is inserted into *pathList*. Paths to other modules can be added in the same way.

- Create a %Library.ListOfDataTypes instance.
- Insert a path for each module that contains a class you will use.
- Add the path list to the Gateway object with Gateway.**%AddToCurrentClassPath()**.

Create a class path list and add to the Gateway object

```
set pathList = ##class(%ListOfDataTypes).%New()
do pathList.Insert("C:\Project\demo\PersonDemo.py")
write !,"Added module: "_pathList.GetAt(1)

do gateway.%AddToCurrentClassPath(pathList)
```

See the previous section for the code to create and connect the Gateway object.

2.2.1 Specifying a Path within a Package Structure

When a module is part of a package, the module path must be specified with dot notation starting at the top level package directory. For example, the path to module PersonDemo.py (listed in "The demo Package") could be specified in either of two ways:

- Standard path notation if treated as a module: `C:\Dev\demo\PersonDemo.py`
- Dot notation if treated as part of package demo: `C:\Dev\demo.PersonDemo.py`

2.3 Creating and Using Proxy Objects

All proxy objects are instances of %Net.Remote.Object. They are created by calling **%New()**, specifying the gateway connection to use and the Python class to be imported. Both the proxy object and the Python object are instantiated when this call is made:

```
set proxy = ##class(%Net.Remote.Object).%New(gateway,className)
```

The arguments for this call are:

- *gateway* — a %Net.Remote.Gateway object connected to an External Python Server (as demonstrated previously in "Creating a Gateway Connection").
- *className* — the fully qualified class name of the Python class. For example, the full name for the Person class (listed in "The demo Package") would be:
  - `PersonDemo.Person` if PersonDemo.py is an unpackaged module.
  - `demo.PersonDemo.Person` if PersonDemo.py is part of package demo.

In the following example, the Gateway object is connected to the PyGate external server. Once the *person* proxy object is created, method *displayPerson()* and property accessors *name* and *age* are available just as they would be in the corresponding Python object:

ObjectScript: creating and using a proxy object

```
  set person = ##class(%Net.Remote.Object).%New(gateway,"PersonDemo.Person")

  write !,"Default values: ",person.displayPerson()

  set person.name="George"
  set person.age=99
  write !,"Changed values: ",person.displayPerson()
  write ", "_person.company.displayEmployer(),!
```

Prints:

```
  Default values: name: Tom, age: 5
  Changed values: name: George, age: 99, employer: Intersystems
```

2.4 The demo Package

The PersonDemo.py module

```
from Company import Company

class Person:

  def __init__(self):
    self._name = "Tom"
    self._age = 5
    self.company = Company()

  @property
  def name(self):
    return self._name

  @name.setter
  def name(self, newName):
    self._name = newName

  @property
  def age(self):
    return str(self._age)

  @age.setter
  def age(self, newAge):
    self._age = newAge

  def displayPerson(self):
    return "name: " + self.name + ", age: " + self.age
```

The Company.py module

```
class Company:

  def __init__(self):
    self._employer = "InterSystems"

  def displayEmployer(self):
    return "employer: " + self._employer
```

3
Managing External Python Servers

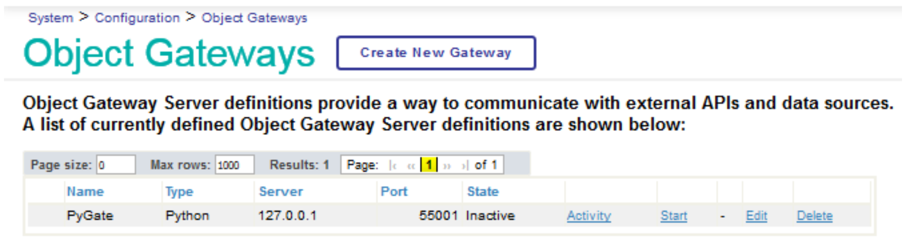
An External Python Server is a Python process that uses the InterSystems gateway connection protocol to communicate with a monitor process running in InterSystems IRIS (see "Introduction to Gateway Connections" for details). This chapter provides a detailed description of how to control gateway connections and create named External Python Server definitions.

- Controlling Gateway Connections — describes the Management Portal page used to start, stop, and monitor gateway connections.
- Configuring External Python Servers — describes the Management Portal form used to define named collections of External Python Server configuration settings.

3.1 Controlling Gateway Connections

The Management Portal provides a page that displays the current status of all defined external servers. It allows you to start or stop external servers, and to control and monitor gateway connections. To open the this page, go to System Administration > Configuration > Connectivity > Object Gateways.

The example shown in the following illustration displays information and controls for a previously created External Python Server definition named PyGate:



Object Gateway Server definitions provide a way to communicate with external APIs and data sources.
A list of currently defined Object Gateway Server definitions are shown below:

Page size: 0 Max rows: 1000 Results: 1 Page: 1 of 1

Name	Type	Server	Port	State					
PyGate	Python	127.0.0.1	55001	Inactive	Activity	Start	-	Edit	Delete

The PyGate External Python Server definition is a named collection of configuration settings used to construct the Python command that will start an External Python Server process (see "Configuring External Python Servers" for information on how the PyGate definition was created).

The display line for PyGate provides the following information:

- PyGate is the unique name for this collection of configuration settings. A named configuration cannot be used to start more than one external server instance at a time.
- *Type* indicates that this definition is for external servers running under Python.

- *Server* is the IP address or name used to connect to the host machine running Python. This example uses the local host address, indicating that Python and the external server run on the same machine as InterSystems IRIS.
- *Port* is the unique port number used whenever InterSystems IRIS establishes a gateway connection to the PyGate external server. No two gateway connections can use the same port simultaneously.
- *State* indicates whether or not a PyGate External Python Server process is currently running.

This display also provides the following controls:

- *Activity* is a link to a page that displays detailed information about all PyGate external server instances that have been started or stopped since InterSystems IRIS was started.
- *Start* will instantiate a PyGate external server and display detailed startup information about the process. If the external server is already running, a *Stop* command will be displayed instead.
- *Edit* opens a form that allows you to change any configuration option except the name PyGate.
- *Delete* allows you to delete the PyGate external server definition.

Note: **"Object Gateways" and External Python Servers**

The name "Object Gateway" is an older term used for external server definitions and gateway connections. Future releases of InterSystems IRIS will remove this Management Portal page and replace it with a completely new External Language Server interface.

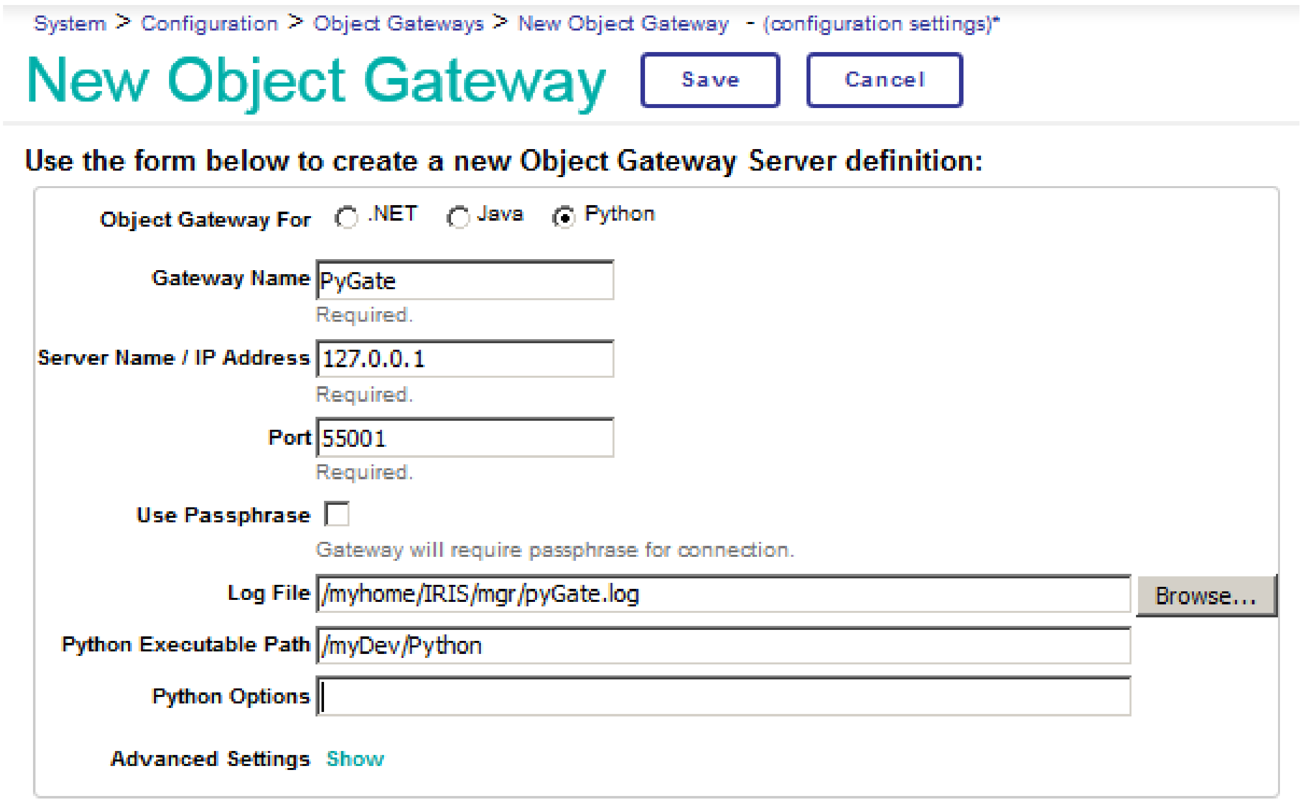
3.2 Configuring External Python Servers

An External Python Server definition is a named collection of configuration settings stored in the InterSystems IRIS database. This information is used to construct the Python command line that starts an external server process, and is also used when InterSystems IRIS attempts to establish a gateway connection to that process. Each external server definition has a unique name, and a definition cannot be used for more than one external server at a time.

The following example creates an External Python Server definition named PyGate. To start, click the `Create New Gateway` button on the Object Gateways page (System Administration > Configuration > Connectivity > Object Gateways).

This form is used to define External Language Servers for all languages (Java and .NET as well as Python). Click the `Python` radio button at the top of the form to display the fields intended specifically for Python external servers.

Figure 3–1: Creating an External Python Server definition



System > Configuration > Object Gateways > New Object Gateway - (configuration settings)*
New Object Gateway
Save
Cancel
Browse...
Show

Use the form below to create a new Object Gateway Server definition:

Object Gateway For	○ .NET ○ Java [✓ ● Python]
Gateway Name	PyGate (Required.)
Server Name / IP Address	127.0.0.1 (Required.)
Port	55001 (Required.)
Use Passphrase	☐ Gateway will require passphrase for connection.
Log File	/myhome/IRIS/mgr/pyGate.log
Python Executable Path	/myDev/Python
Python Options	
Advanced Settings	Show

Settings used by all external servers

- `Object Gateway For [.NET|Java|Python]` *Required.*

  Click `Python` to display the fields required for Python external servers.

- `Gateway Name` *Required.*

  Unique name that you assign to this server definition. The name is used as the database identifier for the stored definition, and cannot be changed after the definition has been saved.

- `Server Name / IP Address`

  IP address or name of the host machine where the external server will run. The default is `"127.0.0.1"` (the local machine). Specify `"0.0.0.0"` to listen on all IP addresses local to the machine (127.0.0.1, VPN address, etc.).

- `Port` *Required.*

  Unique TCP port number for communication between the external server and InterSystems IRIS. No two gateway connections can use the same port simultaneously.

- `Use PassPhrase`

  If this property is checked, InterSystems IRIS will require a security passphrase to connect to the external server.

- `Log File`

Full pathname of the log file on the host machine. External server messages are logged for events such as opening and closing connections to the server, and for error conditions encountered when mapping Python classes to proxy classes. Due to performance considerations, this optional property should only be used for debugging.

Settings that apply only to Python external servers

- `Python Executable Path`

  Fully qualified filename of the Python executable that will be used to run the external server. This setting is optional if there is a default Python interpreter that can be used without specifying its location.

- `Python Options`

  Optional property that defines extra command line settings to be included when constructing the Python command that starts the external server.

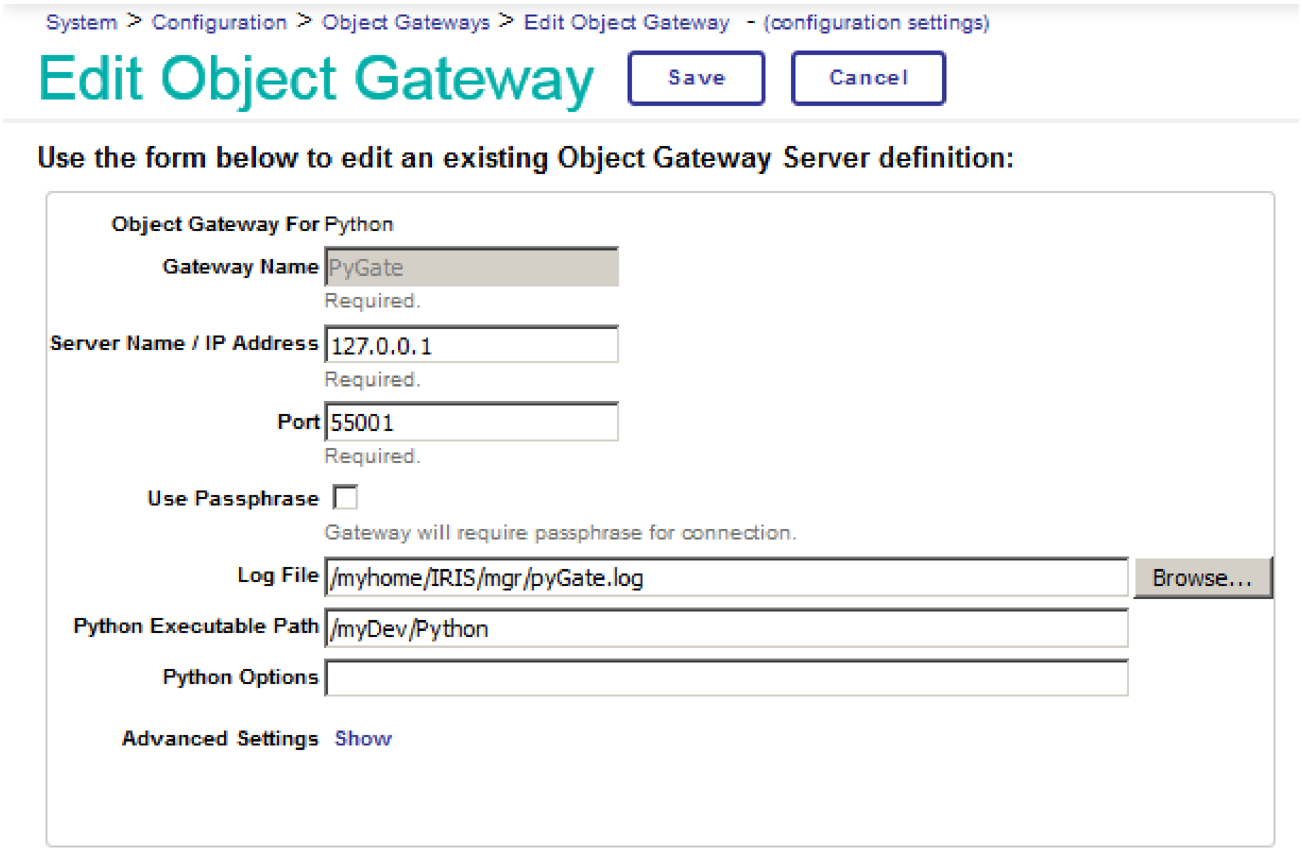
Advanced Settings

The `Advanced Settings` section is relevant only when using gateways in Interoperability Productions (currently supported only for Java and .NET).

Click *Save* to store the external server definition in the database. The `Gateway Name` is the unique identifier for the definition, and cannot be changed after the definition has been saved.

An existing external server definition can be edited by clicking the *edit* link for that definition on the Object Gateway page. This form offers the same fields as the previous example except that `Gateway Name` cannot be changed.

Figure 3–2: Editing an existing External Python Server definition


System > Configuration > Object Gateways > Edit Object Gateway - (configuration settings)
Edit Object Gateway
Save
Cancel
Browse...
Advanced Settings Show

Use the form below to edit an existing Object Gateway Server definition:

Object Gateway For	Python
Gateway Name	PyGate Required.
Server Name / IP Address	127.0.0.1 Required.
Port	55001 Required.
Use Passphrase	☐ Gateway will require passphrase for connection.
Log File	/myhome/IRIS/mgr/pyGate.log
Python Executable Path	/myDev/Python
Python Options	