Python Integration

Quick video tutorial: Using Python with TK Data Access, Using Python with Script Task, Using Python with Code Behind (no audio)

Overview

The programming in many of your projects will consist of C# or VB.Net 100% managed code that is designed to run in the Microsoft .NET framework.

FactoryStudio now includes Python as a programming language you can use in Code Behind, Scripts, Tasks, and interactively via external Python code.

Python is an interpreted, high-level, general purpose language. It is a popular language for machine learning, which is useful for things like Predictive Maintenance algorithms.

Factory Studio can use any version of Python, 3.x or the older and past end-of-life 2.x.

Users can use Python in three different ways:

- Using the Script>Task for Python: Execute Python code (.PY) and using the Script Task interface to set and get parameters (no code needed in the Studio side).
- Python namespace (Python for .Net): Create .Net code using the Python namespace to interact with the Python objects.
- Python using TKDataAccess toolkit: Create Python code using the TKDataAccess.py (provided by us) to interact with the Studio projects.

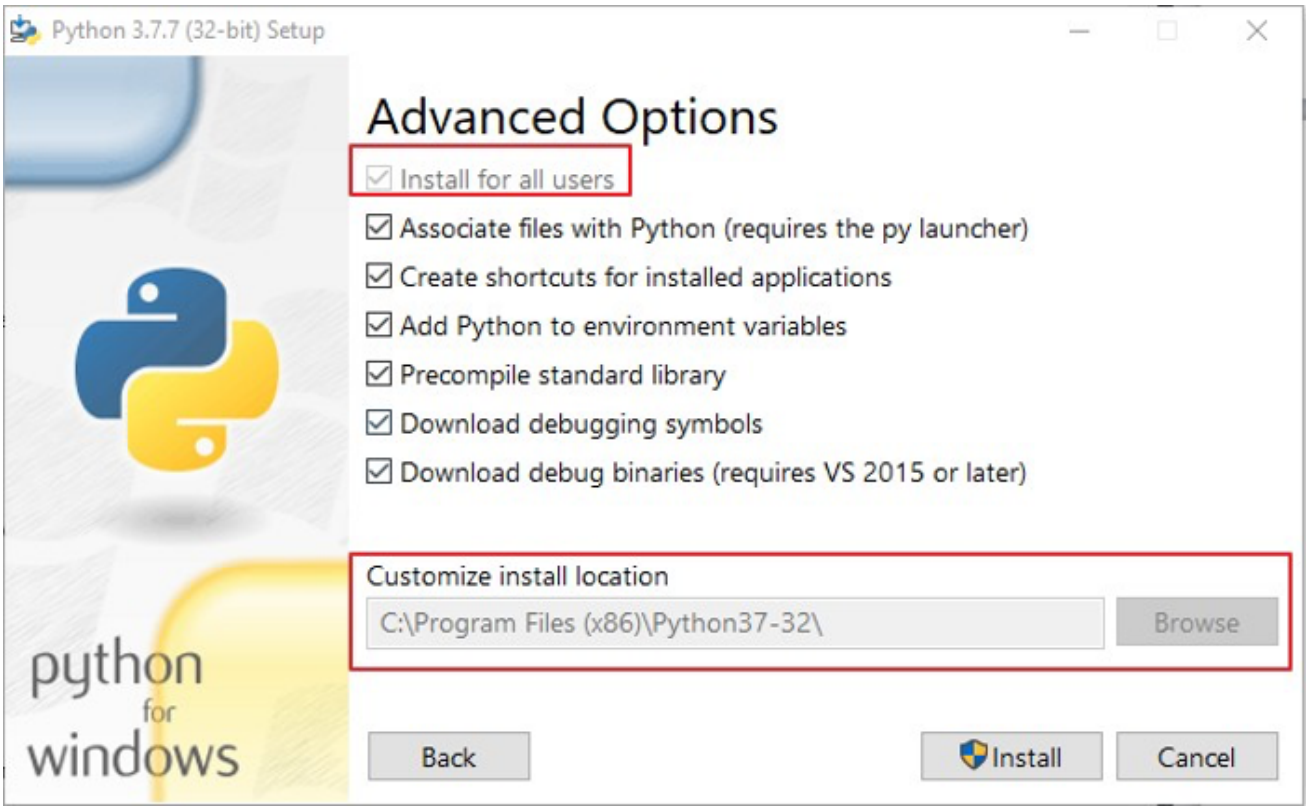
System Requirements

Python Interpreter

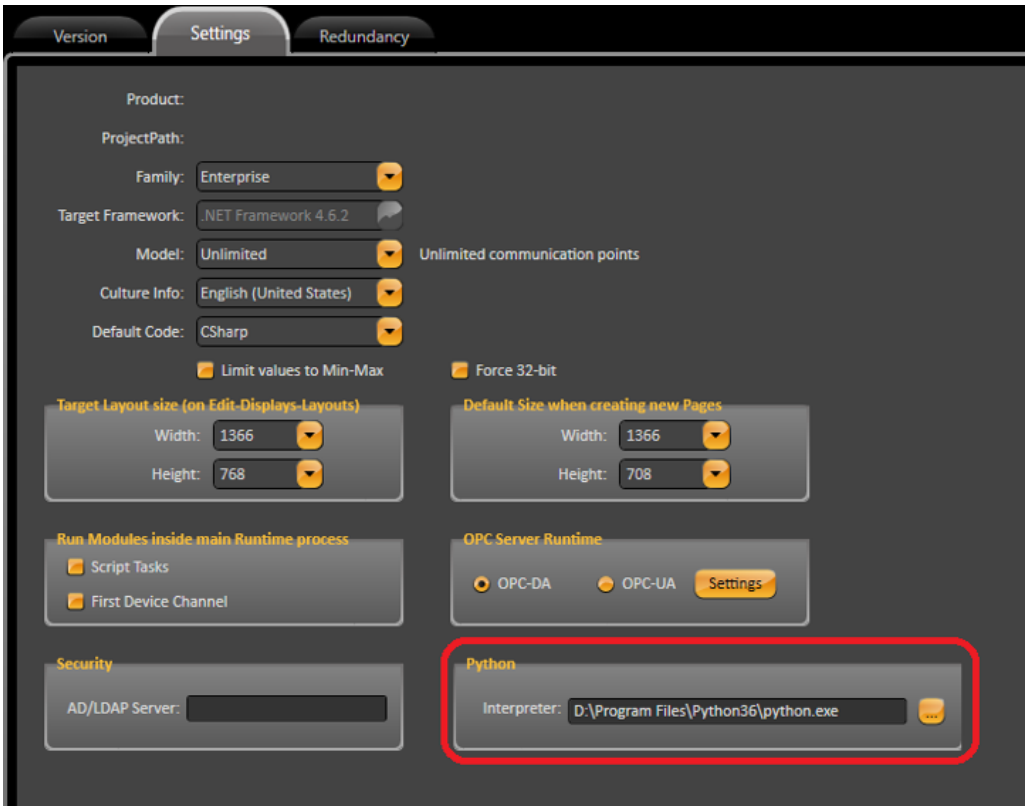
The first step to use Python programming is to include a link to the version of the Python interpreter you want to use for your project.

You can download Python here.

During the installation of the Python Engine, we strongly recommend choosing the option to “Install for all users”.



Now you need to add the interpreter to your project. Navigate to **Info** > **Settings tab** and search for the Python field. Click on the button, browse to find the installed Python Engine, and select the *python.exe* file.



Python for .Net

Python for .NET is a package that gives Python programmers nearly seamless integration with the .NET Common Language Runtime (CLR) and provides a powerful application scripting tool for .NET developers.

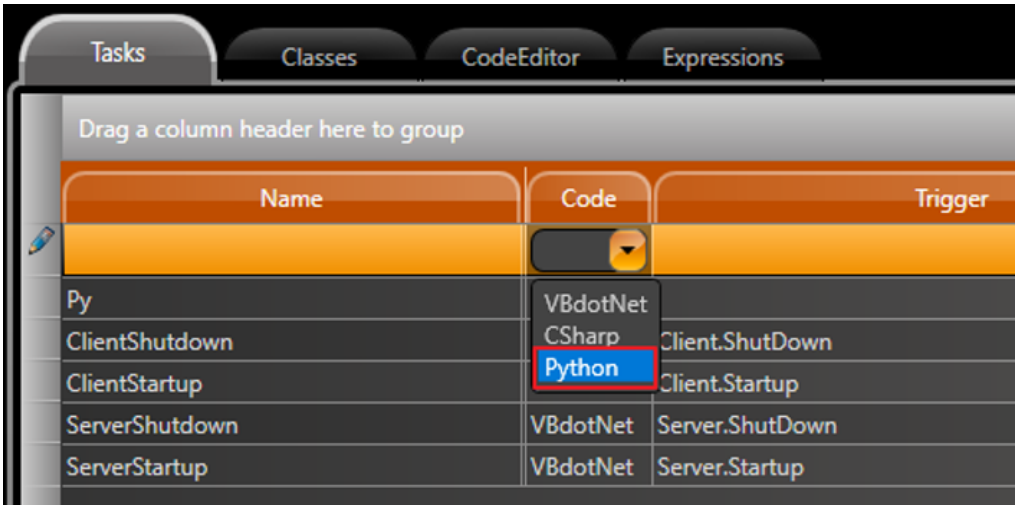
Python for .NET allows Python code to interact with the CLR and may be used to embed Python into a .NET application. The installation files and documentation are available here.

Check to see whether your Python and Windows versions are 32-bit or 64-bit before you download Python for .NET. This is only required if you intend to use the Python namespace in Studio Scripts and Display CodeBehinds.

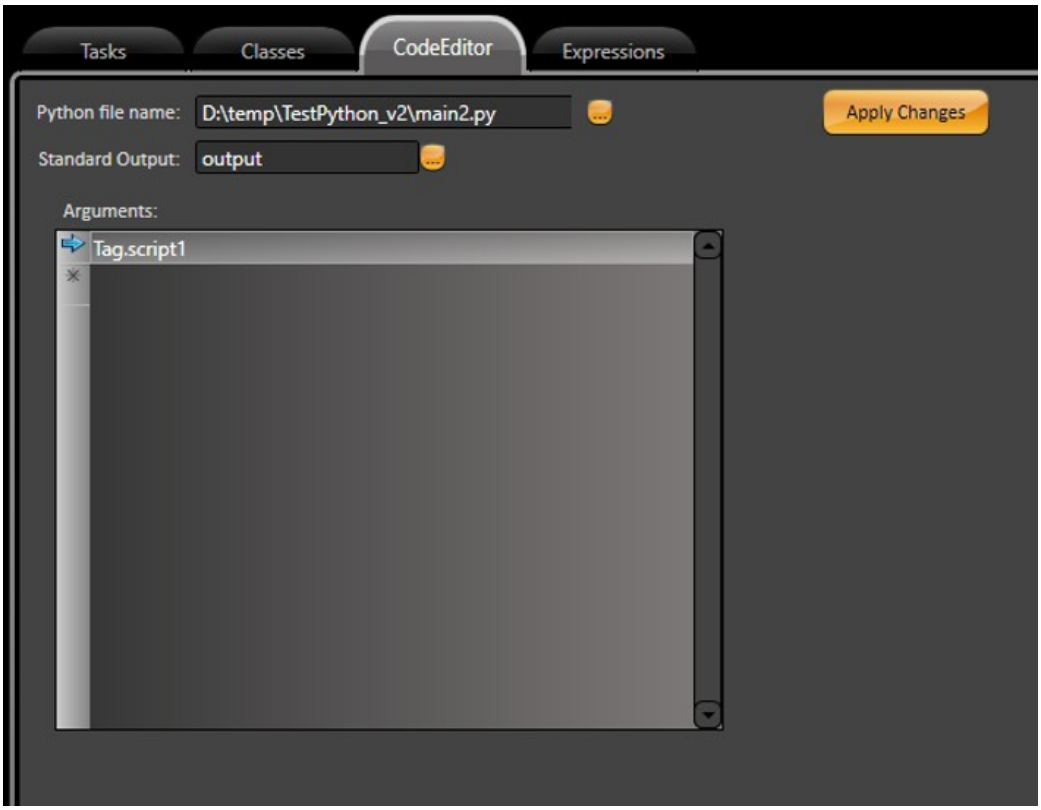
Creating Scripts in Python

To create scripts based on the Python programming language, navigate to **Scripts** > **Tasks**, select a new, blank row, and double-click on the **Code** column . After clicking Code, a combobox will appear. Select Python from the combobox.

> ⚠ Once a new task is created, the language type cannot be altered through the **Code** column.



After creating a new script task, you need to edit it in the **CodeEditor** tab.



The configurations settings are detailed below:

- **File Name**: Name of .py file
- **Standard Output**: Tag name that will receive all output printed via **print**
- **Arguments**: List of arguments. This parameter list will be passed as arguments for the **Python Engine** (python.exe)

After making any changes, click the **Apply Changes** button.

Python Namespace

The Python namespace can be used in any script editor (Tasks, Classes, or CodeBehind) inside your project environment. To use the Python namespace, you simply need to install the **Python.NET** package, available on github.

The Python namespace provides several .Net methods that interact with Python codes and objects. See some of those methods below:

- Run a .py file using Python for .NET

```
string ExecutePyFile(string pyFileName, Dictionary<string, object> locals = null, bool keepValuesAsPython =
false)

string pyFileName = Python file
Dictionary<string, object> locals = Local variables inside Python code. Default is null.
bool keepValuesAsPython = Keep retuned values as Python objects or convert to .NET objects. Default is false.
string returns = If success return null else string contains error.
```

- Run a Python code using Python for .NET

```
string ExecuteCode(string code, string workingDirectory = null, Dictionary<string, object> locals = null, bool
keepValuesAsPython = false)

string code = py file name
string workingDirectory = Working directory. It will be added in 'sys.path'
Dictionary<string, object> locals = Local variables inside Python code. Default is null.
bool keepValuesAsPython = Keep retuned values as Python objects or convert to .NET objects. Default is false.
string return = If success return null else string contains error.
```

- Convert a Python value to a .NET value

```
public static object FromPython(object value)

object value = Python value
object returns = NET value
```

- Copy a Python object to a tag (Array or User Template)

```
public static void CopyPythonObjectToTag(object source, string tagName)

object source = Python object.
string tagName = Tag Array or User Template.
```

- Copy a tag (Array or User Template) to a Python object

```
public static void CopyTagToPythonObject(string tagName, object target)

string tagName = Tag Array or User Template.
object target = Python object.
```

- Create a Python object from a Python class

```
public static object CreatePythonObjectFromPyFile(string pyFileName, string className, object[] parameters =
null, string tagName = null)

string pyFileName = Python file name containg the definition of Python class.
string className = Python class name.
object[] parameters = Parameters for Python class while creating Python object.
string tagName = Tag name (Optional, Tag Array or User Template). If tag exists then copy all values to new
Python object.
object returns = Reference to new Python object.
```

- Get all attributes of a Python object

```
public static IDictionary<string, object> GetAttributesPythonObject(object pythonObject, bool
keepValuesAsPython = false)

object pythonObject = Python object/
bool keepValuesAsPython = Keep retuned values as Python objects or convert to .NET objects. Default is false.
IDictionary<string, object> = Dictionary contains attributes (name and value).
```

- Set a new value for attributes of a Python object

```
public static void SetAttributesPythonObject(object pythonObject, IDictionary<string, object> dic)

object pythonObject = Python object.
IDictionary<string, object> = Dictionary contains attributes (name and value) for setting.
```

- Dump a python object to a string to send it to a TraceWindow

```
public static string DumpPythonObjectToString(object pythonObject)

object pythonObject = Python object.
string returns = Dump information of object.
```

> If you need to install other Python modules and libraries (such as numpy, pythonnet, matplotlib, etc.), you must install them in the same location as Python Engine (python.exe).

> All of the methods listed above are disabled for Mono projects and HTML5 displays.

TKDataAccess.py

You can create code in the Python environment and use the TKDataAccess.py file to interact with the projects.

> Tatsoft provides the TKDataAccess.py file. When you use it, you need to make sure it is installed in the same folder as FactoryStudio.

Below are some methods from TKDataAccess.py that you can use:

- Open a connection with the server

```
Connect(runtimeHostAddress, userName, password):

runtimeHostAddress  =  IP address or server name
userName  = User name.
password = Password
```

- Get a server connection status

```
GetConnectionStatus ()
```

- Check your script's connection to the server

```
IsConnected ()
```

- Disconnect from Server

```
Disconnect()
```

- Set a flag waiting value from server

```
SetSyncFlag(flag):

flag = True wait value from server, false does not wait value from server.
```

- Retrieve a current value

```
GetObjectValue(name)

name = TagName
```

- Set a new value for an object

```
SetObjectValue(name, newValue)

name = TagName
newValue = new value to set in the tag.
```

- Execute a method from a remote ScriptClass

```
ExecuteClassMethodOnServer(className, methodName, parameters)

className = name of the class in the remote project.
methodName = name of the method in the remote class.
parameters = if any, necessary to the invoke the remove method.
```

Example

The sections below contain different ways to use Python in a project.

Using Namespace in CodeBehind

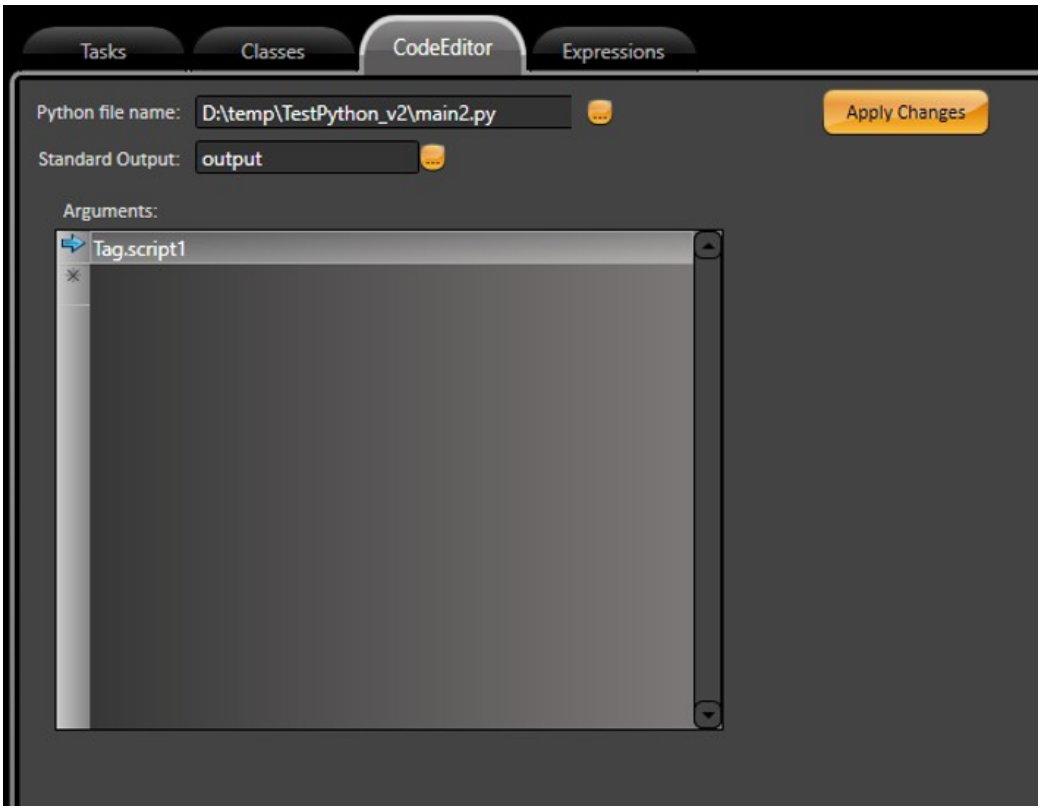
In this example, there are two input parameters called **val1** and **val2** that will be summarized and the result will be stored in the **result** variable.

The code that executes this action is presented below.

```
//Defining where locate the .py files that will use
string  pyDefinition  =  @Info.GetExecutionFolder()  +  @"\Calc\algorithm.py"; string pyWorkingFolder = @Info.
GetExecutionFolder();

try
{
string error;
Dictionary<string,  object>  param  =  new  Dictionary<string,  object>();

       //Defining the used imports from Python
string imports = "";
imports += "import sys" + Environment.NewLine;
imports += "from Calc.algorithm import Algorithm" + Environment.NewLine;

      //Creating a .Net object from a Python object
object algorithm =Python.CreatePythonObjectFromPyFile(pyDefinition, "Algorithm", null);
//Creating a .Net object from a Python object
object val1 = Python.ToPython(@Tag.val1);
object val2 = Python.ToPython(@Tag.val2);

      //Setting the parameters with the .Net objects that will be used to execute the Python code
param.Clear(); param.Add("algorithm", algorithm); param.Add("val1", val1);
param.Add("val2", val2);

      //Call method to execute Python code
error  =  Python.ExecuteCode("result  =  algorithm.Sum(val1,  val2)",  pyWorkingFolder,  param,  true);

if (!string.IsNullOrEmpty(error))
throw new Exception(error);

       //Set .Net object with result Python object , return of algorithm.Sum Python method
object result = param["result"];

//Copy .Net object to Tag
Python.CopyPythonObjectToTag(result, @Tag.result.GetName());
}
catch (Exception ex)
{
@Info.Trace("Python: " + (ex.InnerException == null ? ex.Message : ex.InnerException.Message));
}
```

Using Tasks

In this scenario, we configure a task for the Python language. In the **Python file name** field, you need to set the Python file that will be executed. In this example, we used the **Main2.py**.



In the **Standard Output** field, we selected a tag called **output.** This tag type must be text. In the **Arguments** field, we selected another type of tag called **s cript**.

Using the **print** method, the Python file called **Main2.py** retrieves the input data and outputs its value inside a string. The **sys.argv** will receive the **Tag. script** and the **output** tag will receive all the values from the **print()** method.

```
import sys

value = sys.argv[1] print("Value: " + value) print("That's  all  folks!")
```

Using TKDataAccess.Py

In this example, you need to call a file named **Main.py**, which contains code that copies the content from **tag1** (source) to **tag2** (target).

The Python code using the TKDataAccess.py module in **Main.py** is described below:

```
import sys
from Extensions.TKDataAccess import TKDataAccess dataAccess = TKDataAccess()
connectionStatus = dataAccess.Connect("127.0.0.1:3101", "guest", "") print("Connection: " + connectionStatus)

if dataAccess.IsConnected() :
ret  =  dataAccess.GetObjectValue("Tag.tag1") print("Value: " + str(ret)) dataAccess.SetObjectValue("Tag.
tag2",  ret)

dataAccess.Disconnect()
```

Then, you need to create the code that is shown below or create a Python Script Task that executes the **Main.py** file, which contains the calling for TKDataAccess shown above. So here, we will use the Python namespace as previously described.

The code is shown below.

```
try
{
string  error  =  Python.ExecutePyFile(@Info.GetExecutionFolder()  +  @"\Main.py",  null,  true); if (!string.
IsNullOrEmpty(error))
throw new Exception(error);
}
catch (Exception ex)
{
@Info.Trace("Python: " + (ex.InnerException == null ? ex.Message : ex.InnerException.Message));
}
```


To use the Python namespace, you need to install Python for .NET.