Python and NET Integration Example

Combine Python and .NET in one solution.

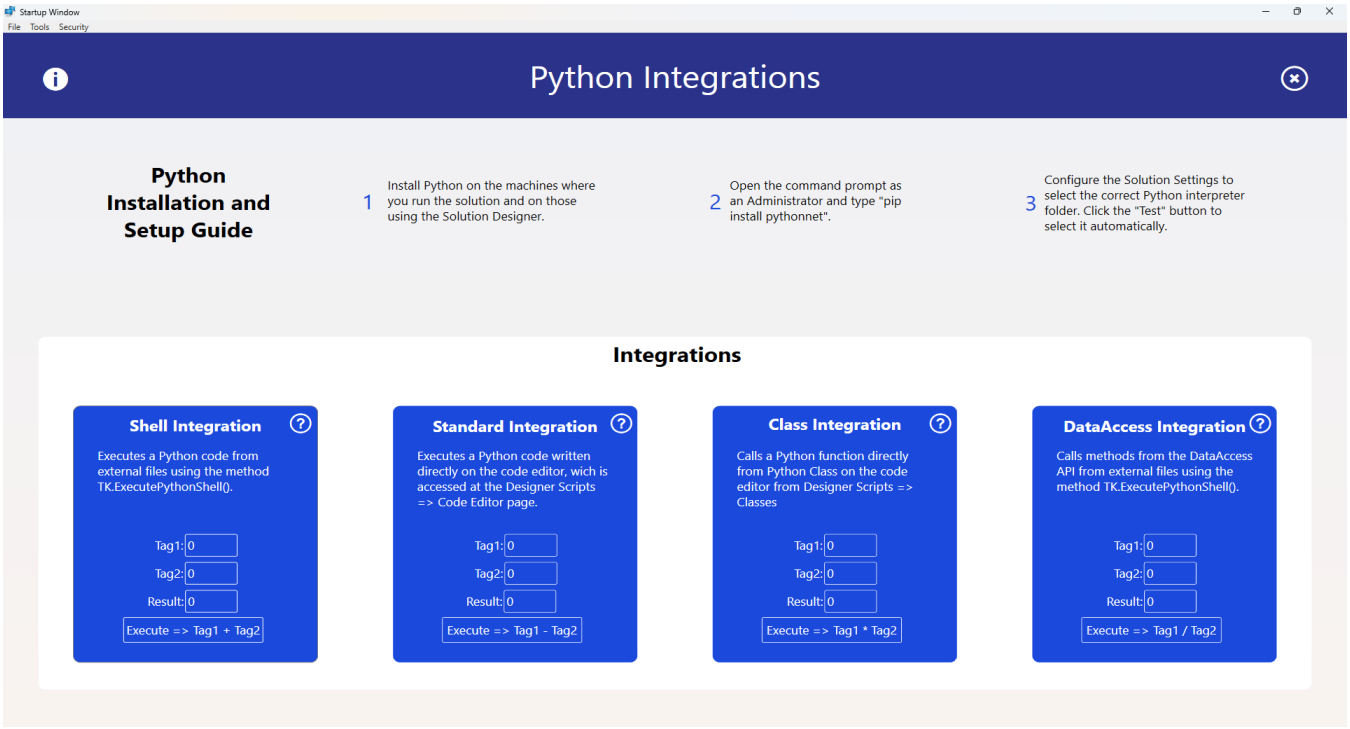
This example demonstrates the integration of Python scripts in the FrameworX environment.

Download the Solution Example:

- **Solution file**: Python and NET Integration Example.dbsln
- **Python file**: ExternalSum.py
- **Python file:** ExternalSumDataAccess.py

Summary

This example demonstrates the integration of Python scripts in the FrameworX environment, showcasing how various Python methods and external scripts can be triggered using tags and buttons in the user interface.



Requirements

The Python Shell integration requires Python releases 3.7 onwards and the installation of Python.NET..

Setup Steps:

1. Install Python and Python.NET in the machines where you run the application and on those using the Solution Designer.
2. Configure the Solution Settings to select the appropriate Python shell folder.

Each solution can specify its own Python interpreter, facilitating the development and maintenance of different solutions that may require different Python versions.

You can download Python here.

Check the Use Admin Privileges, and the option to Customize the installation.

? Unknown Attachment

During the installation of the Python Engine,select the option to “Install for all users”.

? Unknown Attachment

if You install python under your local user, you may have issues when running the solution as a service, or deployment the solution for production.

This image can be slightly different to your Python version, but always enable the options equivalent to:

- Add Python.exe to PATH

Specially in the production server.

- Install with admin rights to all users,
- Customize the installation to installation in the Program Files, instead of Users folder

Once you've downloaded, open command prompt as an ADMINISTRATOR and type “pip install pythonnet”. For this work, you'll need internet access. If internet is not available, download and install manually.

Once Python .NET has been installed, you can then start using Python in your solution.

? Unknown Attachment

Setup the Python location in the Solution

Before you can use Python in a solution, you need to setup in that solution the correct location for the Python installation.

This allows distinct Solutions to utilize specific versions of Python.

Go to **Solution** ***Settings*** tab and locate for the Python GroupBox. Click the "**...**" button, navigate to find the installed Python Engine, and select the *pyt hon.exe* file.

? Unknown Attachment

Python Interpreter	
Local	Defines the path for Python installation folder on the local machine.
Server	Defines the path for Python installation folder on the remote server.

Technical Information

This solution demonstrates the four scenarios on how you can use Python in your solution:

- Python Shell
- Python Standard
- Python and .NET Intermixed
- Python DataAccess API

Each integration method handles tag values, external scripts, or Python classes differently, and they are triggered by either tag changes or button presses. Below is a breakdown of each method:

Python Shell Integration

This integration calls external Python files, passing optional arguments to it, and uses he Python Shell to execute the code capturing the output back to the solution.

Shell Integration example

```
# This code call the execution of the external file using Python Shell
# with the optional args defined in this initial section
#
# The macro _ExecutionPath_ is replaced by the path where the solution is set to execute
# Replace that macro by a specific path, or user other built-in macros as nedded
#

arg1 = @Tag.Tag1
arg2 = @Tag.Tag2
result = TK.ExecutePythonShell("_ExecutionPath_ExternalSum.py", [arg1, arg2])
@Tag.Result = result
```

The method TK.ExecutePythonShell() runs the external python interpreter capturing the ouput, and using the result in the code.

Execution: When this tasks is executed, the script `ExternalSum.py` is called, with the values of `Tag.Tag1` and `Tag.Tag2` passed as arguments. The result, captured from the Print output is captures, and it is stored in `Tag.Result(0)`.

Python Shell Version and Tests

When creating Script > Tasks, with this method, the Play Button ? Unknown Attachment in the top toolbar, will execute the Python code directing the printing output to the Designer Output Window.

Python Standard Integration

Edit and run the Python code within the platform, with no needing for external files.

When creating a new Task, or a new Class, select Python for the language.

Example of. Script Task

```
def sub(val1, val2):
    return val1 - val2

result = sub(@Tag.Tag1[1], @Tag.Tag2[1])
@Tag.Result[1] = result
```

This scenario and the previous one are both relying in the Python interpreter to evaluate the Python code. The difference is that in the first one the python file is external to the solution, and in this one the Python code is saved inside the Solution file.

Python and .NET Intermixed

This integration allows C# and VB.Net to call directly methods written and Python, and the Python to consume methods written in .NET languages.

This is demonstrated in the Python.dbsln demo, when the button is pressed, the following C# expression is executed:

```
Script.Class.ClassPython.sum(Tag.Tag[1], Tag.Tag[2]
```

The Script.Class.ClassPython was created in Script > Classes, using the Python language.

```
def main(self): return def sum(self, val1, val2): return val1 + val2 def multiply(self, val1: int, val2:
int): return val1 * val2
```

Python DataAccess API

This method calls external Python scripts that use the DataAccess API, allowing for interaction with the calling solution, or other solutions in execution.

From the solution, using Scripts Tasks, the following code is executed:

```
# This code call the execution of the external file using Python Shell with the optional args defined in
this initial section
#
# The macro ExecutionPath is replaced by the path where the solution is set to execute
# Replace that macro by a specific path, or user other built-in macros as nedded
#

result = TK.ExecutePythonShell("_ExecutionPath_ExternalSumDataAccess.py", [])
```

The external Python application will use the DataAccess API to connected with the running solution, to read and write data directly.

This is the code for the example: ExternalSumDataAccess.py

```
# TKDataAccess is located with the product installation files
#
# If you move this code to another computer, replace the 127.0.0.1 to server you want to connect
# This code can only run on computers where the product is installed
#
# The Macros _ToolkitPath_ and _ExecutionPort_ are resolved automatically before we call this Python code
# If you call the py file directly you need to replace those macros
#
# _ToolkitPath_ should be replaced by the product binaries folder (location of T.Toolkit.DLL and other DLLs)
# _ExecutionPort_ should be replaced by the port number TServer is running, ex.: 3101 (which is dependent on
the ExecutionProfile)

import sys
installPath = "_ToolkitPath_"
sysPath = ";".join(sys.path)
if sysPath.find(installPath) < 0 :
    sys.path.append(installPath)

from TKDataAccess import TKDataAccess

dataAccess = TKDataAccess()
connectionStatus = dataAccess.Connect("127.0.0.1:_ExecutionPort_", "guest", "")

if dataAccess.IsConnected() :

        value1 = dataAccess.GetObjectValue("Tag.Tag1[3]")
        value2 = dataAccess.GetObjectValue("Tag.Tag2[3]")
        ret = value1 / value2
        dataAccess.SetObjectValue("Tag.Result[3]", ret)
        print(str(ret))
        dataAccess.Disconnect()
```

In this section...
