Chapter 84

Using the Adapter for Python

Python is a high-level, easy to use, powerful, interpreted programming language suitable for scripting, as well as complex programming.

The Python standard library, an extensive collection of modules, implements the Python "batteries included" philosophy that gives programmers immediate access to sophisticated and robust capabilities that make it easy to write your own Python functions to be used in WebFOCUS.

The Adapter for Python defines a connection to the Python interpreter for executing user-written Python scripts that generate calculated fields (WebFOCUS Computes). These fields can be used in WebFOCUS Workbooks, WebFOCUS InfoGraphics, charts, reports, dashboards, and WebFOCUS portals.

In this chapter:

Guidelines for Writing Python Scripts to be Used With TIBCO WebFOCUS

The following Python script, named arithmetic_example.py contains a function named adder that adds two numbers and conforms to the requirements described in this section.

```
# arithmetic_example.py

import csv


def adder(csvin, csvout):

    with open(csvin,  'r', newline='') as file_in,\
         open(csvout, 'w', newline='') as file_out:

        fieldnames = ['addition']

        reader = csv.DictReader(file_in, quoting=csv.QUOTE_NONNUMERIC)

        writer = csv.DictWriter(file_out, quoting=csv.QUOTE_NONNUMERIC,
                                fieldnames=fieldnames)
        writer.writeheader()

        for row in reader:
            addition = row['a_number'] + row['another_number']

            writer.writerow({'addition': addition})
```

A Python script must conform to the following requirements in order to be compatible with WebFOCUS.

- **csv module requirement.** The script must import the csv module because WebFOCUS sends data to the Python script using an automatically generated, temporary .csv file. The name of the file is stored in a global Python variable named *csvin*. The global Python variable *csvout* contains the name of the temporary .csv results file to be returned to WebFOCUS. WebFOCUS sets the values for csvin and csvout, and they should not be changed by the programmer. The temporary files are removed immediately and cannot be viewed by the user. The format of both files is the same. The default delimiter is a comma (,) and cannot be changed. All non-numeric fields are enclosed in double quotation marks.

  The global variables *csvin* and *csvout* are defined by WebFOCUS for reading and writing .csv files.

- **csvin and csvout format parameter.** The script must open csvin and csvout with the following format parameter:

  ```
  newline=''
  ```

If newline='' is not specified, newlines embedded inside quoted fields will not be interpreted correctly, and on platforms that use \r\n line endings on write, an extra \r will be added.

- **reader and writer format parameter.** The following format parameter should be included in the reader and writer statements, whether you are using reader and writer or DictReader and DictWriter, to enclose non-numeric values in double quotation marks and make it clear which values are numeric (they will be converted to floating point values):

  ```
  quoting=csv.QUOTE_NONNUMERIC
  ```

  This indicates that non-numeric (alphanumeric, text, string, and date) values are enclosed in double quotation marks and that numeric values are not. When *csvin* is read, all WebFOCUS numeric values will be automatically converted to Python floating-point numbers. If the WebFOCUS COMPUTE defines the returned field as integer, the decimal point and any decimal places will be truncated. In the csvin file, if a non-numeric field contains the double quotation character, it will be doubled by WebFOCUS. Python will correctly parse this because the Dialect.doublequote format parameter of the Python csv module defaults to *True*.

- **Output.** The function can return multiple output fields. However, each WebFOCUS COMPUTE command can only retrieve a single output field. To retrieve multiple output fields, issue multiple COMPUTE commands. In the call to the PYTHON function, the output argument (the last argument) must match the name of a field in the OUTPUT_DATA segment of the synonym generated for the Python script.

  For example, in the following Master File, the output argument is called ADDITION:

  ```
  SEGMENT=OUTPUT_DATA, SEGTYPE=U, PARENT=INPUT_DATA, $
    FIELDNAME=ADDITION, ALIAS=addition, USAGE=D7.1,
          ACTUAL=STRING,      MISSING=ON,      TITLE='Addition', $
  ```

  Therefore, the last (output) argument name in the call to PYTHON must be ADDITION:

  ```
  COMPUTE Anyname/I5 = PYTHON(synonym_name, anyarg1, anyarg2, ADDITION);
  ```

  The output written to csvout must be a sequence, for example, a list, even for a single field.

  For a list containing a single field, the correct syntax is:

  ```
  writer.writerow([result])
  ```

  The following syntax is incorrect and will return incorrect values for strings and raise an exception for numeric fields:

  ```
  writer.writerow(result)
  ```

- **Functions.** The Python script can contain one or more user-defined functions. When the metadata object (synonym) is created for the script, one of these functions must be selected as the starting point for execution of the script. The definition of the user-written function used as the starting point must contain csvin and csvout as the first positional arguments.

  Note: Because the Python script will be imported, the following Python programming idiom will be ignored.

  ```
  if __name__ == '__main__':
  ```

  However, including it may be useful for testing outside of WebFOCUS.

- **Headers.** Using a header record listing the field names (instead of using positional index numbers) is not required in the sample input data file when creating the synonym for the Python script or when sending data to and retrieving data from the Python script. However, using header records, and, therefore, field names, in the Python script makes it more readable. The following syntax shows a sample of how to implement headers. In the csv.DictReader statement, use of a header record is implied:

  ```
  fieldnames = ['addition']
          reader = csv.DictReader(file_in, quoting=csv.QUOTE_NONNUMERIC)
          writer = csv.DictWriter(file_out, quoting=csv.QUOTE_NONNUMERIC,
                                  fieldnames=fieldnames)
          writer.writeheader()
  ```

  The recommendation is to use header records in the input and output .csv files. If you use sample data without a header, the field names in the generated metadata will be of the form FIELD_1 through FIELD_*n*.

The Adapter for Python also comes with a set or predefined statistical Python functions that you can easily invoke in WebFOCUS.

Prerequisites for Using the Adapter for Python

You can access the complete list of prerequisites when you configure the adapter by right-clicking the adapter name and clicking *Prerequisites*.

On the Linux x64 Intel and Windows platforms, the server includes a fully configured Python 3.6.x release (with required packages) in the EDAHOME etc/python directory. Therefore, a separate Python download, installation, and package installation steps are not required. For these platforms, use the full path EDAHOME etc/python directory displayed in the sample text.

While the EDAHOME etc/python release is the preferred configuration choice, you can still point an external release, if needed (such as for additional packages), provided it conforms to the instructions listed in the Prerequisites page for the adapter.

- Python must be installed on the same machine as the WebFOCUS Reporting Server, using the same bit size (32 or 64-bit).
- The Adapter for Python is available on Windows, z/OS, and Linux.
- The required Python release level is 3.6.x. Do not deselect the pip option in the install.
- The following packages must be installed in the following order (instructions are on the Prerequisites page):
  - numpy.
  - scipy (needs numpy).
  - scikit-learn (needs both of them and will add additional packages).
  - pandas.
- The system variables you may need to set are described on the Prerequisites page for the adapter.

Configuring the Adapter for Python

Configuring the adapter consists of identifying your Python installation directory.

Procedure: How to Configure the Adapter for Python

1. In the Reporting Server Reporting Server browser interface, click *Get Data*.
2. Click *New Datasource*, the right-click *PYTHON*. Note that PYTHON can be found in the Statistics category on the *Available* drop-down list.

   The Add PYTHON to Configuration page opens.
3. Enter the path to your Python installation directory, and click *Test*.

   The following message displays for a successful configuration.

   ```
   Successful test for the Python environment
   ```
4. Click *Configure*

   The Adapter for Python is added to the list of Configured Adapters.

Creating Synonyms for Python Functions

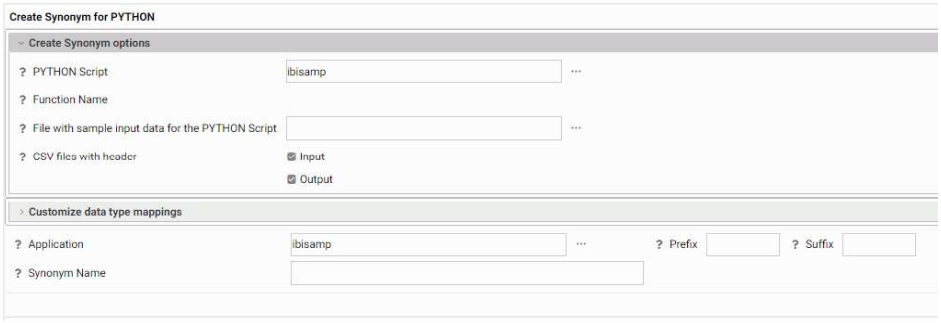
Each Python script used with the Adapter for Python must have a synonym (metadata object) that describes the input fields and output fields of the script. If a Python script contains multiple user written functions, and you want to be able to use more than one function within the script as a starting point, you must create a separate synonym for each function within the script.

The synonym will be created using a sample file that contains only the fields that are input parameters for the script. A few rows of sample data are sufficient for the Adapter for Python to determine the appropriate data types and lengths of the parameters. The sample file must be a .csv file. The data in the file does not have to contain actual data, but it should represent the highest values for numeric fields and the longest lengths for alphanumeric values that will appear the actual data. The Master File will contain the list of input fields and output fields. The Access File will contain information about the script file and sample input file.

Procedure: How to Create a Synonym for a Python Function

1. Right-click PYTHON on the list of configured adapters, and click *Create metadata objects* on the context menu.

   The Create Synonym for Python frame opens, as shown in the following image.



   Note that a metadata object is a synonym. The synonym for a Python function will consist of a Master File (which describes the input fields and output fields needed for running the function) and an Access File (which contains information about the sample data file and the script file).

2. Enter or select values for the following parameters.

   PYTHON Script

   Is the Python script. Enter an application directory name and script name, or click the ellipsis (...) to navigate to an application directory and select a script, then click *OK*. The Python script will have the extension .py.

Function Name

Select the name of the (starting) function in the script file for which to create a synonym.

For example, the Python script named arithmetic_example.py contains the definition for the function adder:

```
def adder(csvin,csvout):
```

Select file with sample input data for the PYTHON Script

Open the file picker (...) to select the application directory and file that contains the sample data for creating the synonym. Click *OK*.

This file is used to determine the field names, data types, and lengths for the data sent to the Python script in *csvin*. If the sample file has no header record, the field names will be FIELD_1 through FIELD_*n*.

CSV files with header

If the input .csv file does not have a header row, uncheck *Input*. If the output .csv file should not have a header row, uncheck *Output*. The header requirements are contained in the function code.

For example, in the following sample code the input file contains no header record (fieldnames) and the fieldnames object is defined at runtime using the fieldnames argument for the reader. The output file will contain a header record, as defined in the fieldnames argument for the writer, and is written to the file using the statement writer.writeheader():

```
with open(csvin,  'r', newline='') as file_in, \
         open(csvout, 'w', newline='') as file_out:
reader = csv.DictReader(file_in, fieldnames=['input_field'],
                        quoting=csv.QUOTE_NONNUMERIC)
writer = csv.DictWriter(file_out, fieldnames=['output_field'],
                        quoting=csv.QUOTE_NONNUMERIC)
writer.writeheader()
```

Application

Enter the name of the application directory in which to create the synonym, or click the ellipsis (...) to navigate to an application directory, then click *OK*.

Synonym Name

Enter a name for the resulting synonym, or accept the default name.

3. Click the Create Synonym button on the ribbon.

The synonym is created in the specified application directory.

Note: You can generate sample Python scripts and data files in the Reporting Server Reporting Server browser interface. Create an application directory to contain the sample files, right-click the application folder, point to *New*, and click *Tutorials*. On the Tutorials page, select the *WebFOCUS - Retail Demo* tutorial, make sure *Create Python Example* is checked and that *Large* or *Medium* is selected for *Tutorial Data Volume Limit*. Click *Create* to create the demo files.

Running a User-Written Python Script

To run a Python script, you call the WebFOCUS PYTHON function. The arguments you supply to the PYTHON function consist of the synonym for the Python script, the input fields, and the output field.

Syntax: How to Run a User-Written Python Function

```
PYTHON([app/]synonym, input1 [, input2 ...], output)
```

where:

[app/]synonym
Is the application and synonym name for the Python script.

input1 [, input2 ...]
Are the input arguments.

output
Is the output argument. This argument must match the name of a field in the OUTPUT_DATA segment in the Master File.

Example: Running a Python Script

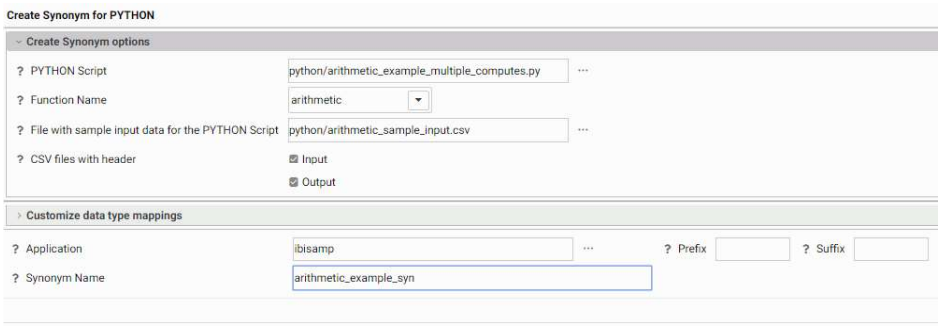
The following is the arithmetic_example_multiple_computes.py Python script, which calculates four output fields, ADDITION, SUBTRACTION, MULTIPLICATION, and DIVISION in the function named *arithmetic*. All of the files for this example reside in an application directory named *python*.

```
# arithmetic_example_multiple_computes.py

import csv
import time

def arithmetic(csvin, csvout):

    with open(csvin,  'r', newline='') as file_in,\
         open(csvout, 'w', newline='') as file_out:

        fieldnames = ['addition', 'subtraction',
                      'multiplication', 'division']

        reader = csv.DictReader(file_in, quoting=csv.QUOTE_NONNUMERIC)

        writer = csv.DictWriter(file_out, quoting=csv.QUOTE_NONNUMERIC,
                                fieldnames=fieldnames)
        writer.writeheader()

        for row in reader:
            addition       = row['a_number'] + row['another_number']
            subtraction    = row['a_number'] - row['another_number']
            multiplication = row['a_number'] * row['another_number']
            division       = row['a_number'] / row['another_number']

            writer.writerow({'addition':       addition,
                             'subtraction':    subtraction,
                             'multiplication': multiplication,
                             'division':       division})
```

The .csv file with the sample data, arithmetic_sample_input.csv, has a header record and two data records to be used to determine the data types and lengths for the input arguments.

```
"a_number","another_number"
1,1
100000,100000
```

The synonym creation frame for this script is shown in the following image.



The generated Master File (arithmetic_example_syn.mas) follows:

```
FILENAME=ARITHMETIC_EXAMPLE_SYN, SUFFIX=PYTHON  , $
  SEGMENT=INPUT_DATA, SEGTYPE=S0, $
    FIELDNAME=A_NUMBER, ALIAS=a_number, USAGE=I11, ACTUAL=STRING,
      MISSING=ON,
      TITLE='a_number', $
    FIELDNAME=ANOTHER_NUMBER, ALIAS=another_number, USAGE=I11,
ACTUAL=STRING,
      MISSING=ON,
      TITLE='another_number', $
  SEGMENT=OUTPUT_DATA, SEGTYPE=U, PARENT=INPUT_DATA, $
    FIELDNAME=ADDITION, ALIAS=addition, USAGE=D10.1, ACTUAL=STRING,
      MISSING=ON,
      TITLE='addition', $
    FIELDNAME=SUBTRACTION, ALIAS=subtraction, USAGE=D5.1, ACTUAL=STRING,
      MISSING=ON,
      TITLE='subtraction', $
    FIELDNAME=MULTIPLICATION, ALIAS=multiplication, USAGE=D15.1,
ACTUAL=STRING,
      MISSING=ON,
      TITLE='multiplication', $
    FIELDNAME=DIVISION, ALIAS=division, USAGE=D5.1, ACTUAL=STRING,
      MISSING=ON,
      TITLE='division', $
```

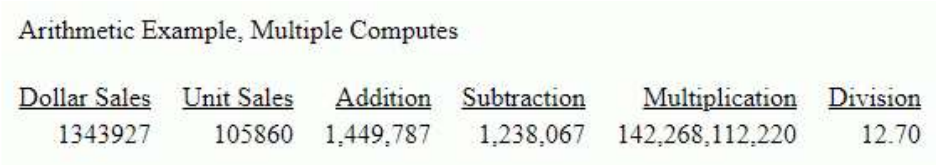
The generated Access File (arithmetic_example_syn.acx) follows:

```
SEGNAME=INPUT_DATA,
  MODNAME=python/arithmetic_example_multiple_computes.py,
  FUNCTION=arithmetic,
  PYTHON_INPUT_SAMPL=python/arithmetic_sample_input.csv,
  INPUT_HEADER=YES,
  OUTPUT_HEADER=YES, $
```

The following WebFOCUS procedure, sales_multiple_computes.fex calls the arithmetic function four times in order to get a value returned for each of the four outputs:

```
-* sales_multiple_computes.fex

TABLE FILE GGSALES
SUM
  DOLLARS
  UNITS

COMPUTE Addition/D7        = PYTHON(python/arithmetic_example_syn,
                            DOLLARS, UNITS, ADDITION);
COMPUTE Subtraction/D7     = PYTHON(python/arithmetic_example_syn,
                            DOLLARS, UNITS, SUBTRACTION);
COMPUTE Multiplication/D16 = PYTHON(python/arithmetic_example_syn,
                            DOLLARS, UNITS, MULTIPLICATION);
COMPUTE Division/D7.2      = PYTHON(python/arithmetic_example_syn,
                            DOLLARS, UNITS, DIVISION);

WHERE RECORDLIMIT EQ  100
HEADING
  "Arithmetic Example, Multiple Computes"
  ""

ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.



Arithmetic Example, Multiple Computes

Dollar Sales	Unit Sales	Addition	Subtraction	Multiplication	Division
1343927	105860	1,449,787	1,238,067	142,268,112,220	12.70