Parameter Passing in Python

For each `in` parameter of a Slice operation, the Python mapping generates a corresponding parameter for the method in the skeleton. In addition, every operation has a trailing parameter of type `Ice.Current`. For example, the `name` operation of the `Node` interface has no parameters, but the `name` method in a Python servant has a `current` parameter. We will ignore this parameter for now.

Parameter passing on the server side follows the rules for the client side. An operation returning multiple values returns them in a tuple consisting of a non-`void` return value, if any, followed by the `out` parameters in the order of declaration. An operation returning only one value simply returns the value itself.

> An operation returns multiple values when it declares multiple out parameters, or when it declares a non-`void` return type and at least one `out` parameter.

To illustrate these rules, consider the following interface that passes string parameters in all possible directions:

Slice

```
interface Example {
    string op1(string sin);
    void op2(string sin, out string sout);
    string op3(string sin, out string sout);
};
```

The generated skeleton class for this interface looks as follows:

Python

```
class Example(Ice.Object):
    def __init__(self):
        # ...

    #
    # Operation signatures.
    #
    # def op1(self, sin, current=None):
    # def op2(self, sin, current=None):
    # def op3(self, sin, current=None):
```

The signatures of the Python methods are identical because they all accept a single `in` parameter, but their implementations differ in the way they return values. For example, we could implement the operations as follows:

Python

```
class ExampleI(Example):
    def op1(self, sin, current=None):
        print sin             # In params are initialized
        return "Done"         # Return value

    def op2(self, sin, current=None):
        print sin             # In params are initialized
        return "Hello World!" # Out parameter

    def op3(self, sin, current=None):
        print sin             # In params are initialized
        return ("Done", "Hello World!")
```

Notice that `op1` and `op2` return their string values directly, whereas `op3` returns a tuple consisting of the return value followed by the `out` parameter.

This code is in no way different from what you would normally write if you were to pass strings to and from a function; the fact that remote procedure calls are involved does not impact your code in any way. The same is true for parameters of other types, such as proxies, classes, or dictionaries: the parameter passing conventions follow normal Python rules and do not require special-purpose API calls.

See Also

- Server-Side Python Mapping for Interfaces
- Python Mapping for Operations
- Raising Exceptions in Python
- The Current Object