Design and implementation of XMLProcessor object for Shahkar

Dave Evans
evansde@fnal.gov

16th February 2004

1 Overview

The `XMLProcessor` is a toolkit for saving Python Objects as XML, and extracting them again to reproduce the state of an object when it is initially saved. There are two parts to this process:

1. **Conversion** Producing an XML Representation of a Python Object in memory.

2. **Extraction** Producing a Python Object in memory from an XML Representation

The `XMLProcessor` Object provides a simple interface to converting objects based on type. Most of the basic python types are supported in the `XMLProcessor` Class, and an interface for registering new User-defined types are provided, so that subclasses of `XMLProcessor` can extend the functionality of the processor to new object types. Nested types are supported in a recursive fashion, so that, for example, a list of dictionaries mapping strings to floats could be converted easily.

2 Concept

The concept behind the `XMLProcessor` code is to provide a simple API to convert python objects into XML to enable them to be saved, and extract the object from teh XML at a later time. This process is illustrated in Fig. ??.

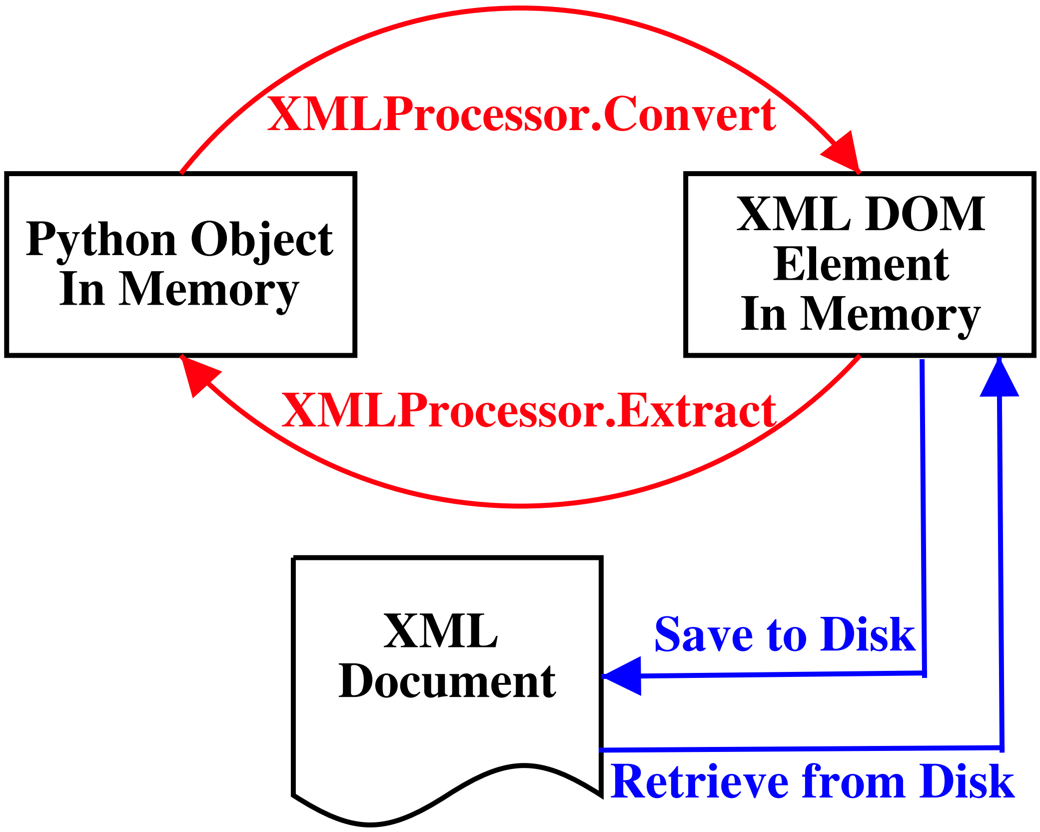

Figure 1: Diagram showing the concepts involved with converting an python object to an XML representation and back again, so that objects can be saved to disk, revived and edited, and saved again etc.

The Conversion and Extraction API is very simple to use and interrogates the object for its type, which decides how the object will be converted. Two exception classes are provided, one for errors during conversion and one for errors during extraction, all other errors are converted to these types to provide a simple API as can be seen in the following examples.

2.1 Conversion

Basic Python types can be converted using an instance of the `XMLProcessor` object as follows:

```
#  Define A simple dictionary object
dict = ‘‘Key’’:’’value’’
#  Create an Instance of XMLProcessor
proc = XMLProcessor()
try:
   #  Convert dict into an XML DOM Object
   elem=proc.Convert(dict)
   #  Print the XML representation of the Object
   print elem.toxml()
except XMLConvertException,ex:
   print ex
```

This Example provides a DOM XML Element containing a representation of the “dict” Dictionary object. The DOM Element is part of the standard python xml.dom.minidom package. XML DOM Elements as returned by Convert can be dumped to the screen by calling print on the return value of the toxml() method. The Element returned by the Convert method can be added to a DOM Document object to be saved as part of an XML File.

2.2 Extraction

A DOM XML Element object that has been converted to xml using the Convert method can be recovered from the XML Element by passing the Element to the processor. The following example continues the above example by extracting the dictionary object from the XML Element produced by Convert.

```
#  Define A simple dictionary object
dict = “Key”:”value”
#  Create an Instance of XMLProcessor
proc = XMLProcessor()
#  elem is an XML DOM Element representing dict
elem=proc.Convert(dict)
try:
   dict2 = proc.Extract(elem)
   print dict2 == dict
except XMLPExtractException,ex:
   print ex
```

3 Python Type Support

Most python types are supported automatically, simple types such as strings, ints and floats are converted into XML, where sequence and mapping types are converted recursively so that each element of the container gets converted. Nested data types are handled automatically, so for instance, a dictionary containing lists will be converted recursively until all data types are converted. Python method references to functions and class methods are also supported, and basic support for converting user defined classes via an InstanceType converter is also provided, although this has some limitations. For Complex Objects, the `XMLProcessor` can be subclassed to handle custom types, as described in §**??**.

Simple Types that are supported by `XMLProcessor` are summarised in Table **??**.

3.1 Limitations

There are some limitations with the conversion/extraction of some types:

MethodType MethodType refers to class methods, and are usually bound to an instance of the class that they are defined in. In the current code references to bound methods are extracted as unbound methods, meaning there is no object instantiated for the method reference. This means that unbound methods created by the extraction process need to be called with

Type	Converter	Extractor
IntType	_IntConverter	_IntExtractor
FloatType	_FloatConverter	_FloatExtractor
StringType	_StrConverter	_StrExtractor
NoneType	_NoneConverter	_NoneExtractor
ListType	_ListConverter	_ListExtractor
TupleType	_TupleConverter	_TupleExtractor
DictType	_DictConverter	_DictExtractor
FunctionType	_FuncConverter	_FuncExtractor
MethodType	_MethodConverter	_MethodExtractor
InstanceType	_InstanceConverter	_InstanceExtractor

Table 1: Table of supported python types and their associated conversiona and extraction methods of `XMLProcessor`.

an instance of the class as the first method, via the __call__ attribute of the method reference.

InstanceType All user defined classes are of python type InstanceType, meaning that they can be very complex objects. InstanceType support saves each of the objects attributes as they appear in its __dict__ table. When InstanceTypes are extracted, and instance of the object is created using its constructor. There is currently no way to pass parameters to the object constructor, so classes which require arguments in their constructor will not work using InstanceType and the `XMLProcessor` needs to be extended to cope with them.

4 Exceptions

5 Extending XMLProcessor

The `XMLProcessor` class provides several methods to facilitate subclassing to handle complex data types that require special support or cannot be handled by the provided native python type converters. Extending the

`XMLProcessor` to handle new objects is performed by subclassing the `XMLProcessor` class to define your own processor. This process is quite simple and consists of defining a type for your own class, a Conversion Method to produce an XML representation and an Extraction Method to rebuild an object from the XML Element. The new type and methods must be Registered with the processor in the constructor of the subclass. Also, the class must provide an xmlType method as described in §??

5.1 Registering New Types

New Types can be registered via the RegisterType method provided by the `XMLProcessor` object. The RegisterType method takes three arguments (In Order):

- The new Type
- A Conversion method associated with that type
- An Extraction method associated with that type

5.2 Requirements On Objects

Object to be passed to the processor for conversion to XML must provide a Type definition, which is usually a string. The Object must also provide a method to return this type, so that the `XMLProcessor` can interrogate it and match it with the appropriate Conversion and Extraction Operators.

A trivial example of this is as follows:

```
#  Define Type of Foo Class Object
FooType = ‘‘FooTypeDefinition’’
#  User Foo Class Definition
class Foo:
   #  Return Type Definition
   def xmlType(self):
       return FooType
```

Since the nature of user defined classes is arbitrary, the `XMLProcessor` should be subclassed to provide functionality for converting user defined objects that cannot be handled by the built in python InstanceType support.