# Package 'XRPython'

October 12, 2022

**Type** Package

**Title** Structured Interface to 'Python'

**Version** 0.8

**Date** 2017-10-22

**Author** John M. Chambers

**Maintainer** John Chambers <jmc@r-project.org>

**Description** A 'Python' interface structured according to the general form described in package 'XR' and in the book ``Extending R''.

**Collate** pythonInterface.R classStructure.R pythonProxyClasses.R

**License** GPL (>= 2)

**Imports** methods, XR (>= 0.7.1), reticulate

**NeedsCompilation** no

**RoxygenNote** 6.0.1

**Repository** CRAN

**Date/Publication** 2017-10-24 22:29:39 UTC

## R topics documented:

---

`XRPython-package` *Functional and Object-based Interface from R to Python*

---

## Description

An Interface from R to Python using, in a very preliminary way, the principles of functional and object-based interface design presented in the reference.

## Author(s)

John M. Chambers

## References

Chambers, John M. (book in progress) *Extending R*

---

`convert` *Convert Proxy Objects between XRPython and reticulate*

---

## Description

Packages XRPython and reticulate both support proxies for Python objects; that is, R objects that are proxies for objects created in Python by evaluations in the respective packages. Function `fromRtclt()` returns the equivalent XRPython proxy object given a reticulate object. Function `toRtclt()` returns the equivalent reticulate proxy object given an XRPython object. Normally, no copying is involved in either direction.

## Usage

```
fromRtclt(obj, .ev = XRPython::RPython())

toRtclt(obj, .ev = XRPython::RPython())
```

## Arguments

| | |
|---|---|
| `obj` | a proxy object, computed in XRPython for `toRtclt` or by reticulate for `fromRtclt` |
| `.ev` | an XRPython evaluator, by default and usually the current evaluator. |

### Functions

- `fromRtclt`: Convert from reticulate to XRPython
- `toRtclt`: Convert from XRPython to reticulate

---

`dict_Python-class` *Proxy Class for Python Dictionaries*

---

### Description

This class is a proxy for ordinary dictionary objects in Python. All the standard Python methods for such objects (e.g., `keys()`) are available, but methods for R functions are not implemented.

### Methods

`clear(..., .ev = XRPython::RPython(), .get = NA)` Python Method: clear() D.clear() -> None. Remove all items from D.

`fromkeys(..., .ev = XRPython::RPython(), .get = NA)` Python Method: fromkeys() dict.fromkeys(S[,v]) -> New dict with keys from S and values equal to v. v defaults to None.

`get(..., .ev = XRPython::RPython(), .get = NA)` Python Method: get() D.get(k[,d]) -> D[k] if k in D, else d. d defaults to None.

`has_key(..., .ev = XRPython::RPython(), .get = NA)` Python Method: has_key() D.has_key(k) -> True if D has a key k, else False

`items(..., .ev = XRPython::RPython(), .get = NA)` Python Method: items() D.items() -> list of D's (key, value) pairs, as 2-tuples

`iteritems(..., .ev = XRPython::RPython(), .get = NA)` Python Method: iteritems() D.iteritems() -> an iterator over the (key, value) items of D

`iterkeys(..., .ev = XRPython::RPython(), .get = NA)` Python Method: iterkeys() D.iterkeys() -> an iterator over the keys of D

`itervalues(..., .ev = XRPython::RPython(), .get = NA)` Python Method: itervalues() D.itervalues() -> an iterator over the values of D

`keys(..., .ev = XRPython::RPython(), .get = NA)` Python Method: keys() D.keys() -> list of D's keys

`pop(..., .ev = XRPython::RPython(), .get = NA)` Python Method: pop() D.pop(k[,d]) -> v, remove specified key and return the corresponding value. If key is not found, d is returned if given, otherwise KeyError is raised

`popitem(..., .ev = XRPython::RPython(), .get = NA)` Python Method: popitem() D.popitem() -> (k, v), remove and return some (key, value) pair as a 2-tuple; but raise KeyError if D is empty.

`setdefault(..., .ev = XRPython::RPython(), .get = NA)` Python Method: setdefault() D.setdefault(k[,d]) -> D.get(k,d), also set D[k]=d if k not in D

`update(..., .ev = XRPython::RPython(), .get = NA)` Python Method: update() D.update([E, ]**F) -> None. Update D from dict/iterable E and F. If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

`values(..., .ev = XRPython::RPython(), .get = NA)` Python Method: values() D.values() -> list of D's values

`viewitems(..., .ev = XRPython::RPython(), .get = NA)` Python Method: viewitems() D.viewitems() -> a set-like object providing a view on D's items

`viewkeys(..., .ev = XRPython::RPython(), .get = NA)` Python Method: viewkeys() D.viewkeys() -> a set-like object providing a view on D's keys

`viewvalues(..., .ev = XRPython::RPython(), .get = NA)` Python Method: viewvalues() D.viewvalues() -> an object providing a view on D's values

---

`from_Python-class` *Class for General Python Class Objects*

---

## Description

The Python side of the interface will return a general object from a Python class as an R object of class "from_Python". Its Python fields (converted to R objects) can be accessed by the $ operator.

## Slots

`serverClass` the Python type.

`module` the Python module, or ""

`fields` the converted versioin of the Python fields; these are accessed by the $ operator.

---

`functions` *Function Versions of Methods for Python Interface evaluators.*

---

## Description

These functions allow application code to invoke evaluator methods for essentially all basic computations. Usually, they access the current Python evaluator, starting one if none exists. For details, see the documentation for the corresponding method, under PythonInterface.

## Usage

```
pythonSend(object, evaluator = XR::getInterface(.PythonInterfaceClass))

pythonEval(expr, ..., evaluator = XR::getInterface(.PythonInterfaceClass))

pythonCommand(expr, ..., evaluator = XR::getInterface(.PythonInterfaceClass))

pythonCall(fun, ..., evaluator = XR::getInterface(.PythonInterfaceClass))

pythonGet(object, evaluator = XR::getInterface(.PythonInterfaceClass))
```

```
pythonSource(file, ..., evaluator = XR::getInterface(.PythonInterfaceClass))

pythonDefine(text, file, ...,
  evaluator = XR::getInterface(.PythonInterfaceClass))

pythonSerialize(object, file, append = FALSE,
  evaluator = XR::getInterface(.PythonInterfaceClass))

pythonUnserialize(file, all = FALSE,
  evaluator = XR::getInterface(.PythonInterfaceClass))

pythonName(object)

pythonShell(..., evaluator = XR::getInterface(.PythonInterfaceClass))
```

## Arguments

| | |
|---|---|
| object | For pythonSend(), an R object, to be sent to Python. For pythonGet(), pythonSerialize() and pythonNqme(), a proxy object for a Python object. |
| evaluator | The evaluator object to use. By default, and usually, the current evaluator is used, and one is started if none has been. |
| expr | A string for a Python expression or command, with C-style fields ("%s") to be substituted for the following arguments, if any. |
| ... | For the evaluation functions: Objects, either R objects to be converted or proxies for Python objects previously computed. For other functions, specialized arguments for the corresponding method. In particular, .get= for controlling whether the computed result should be converted. |
| fun | the string name of the function; a module name must be included in the string if the function has not been explicitly imported from that module. |
| file | A filename or an open connection, for reading or writing depending on the function |
| text | the definition as text (supply argument file= instead to read it from a file) |
| append | should the serializing text be appended to a file; otherwise the file will be truncated on opening. |
| all | should the unserialized object be a list of all serialized objects on the file? |

## Functions

- pythonSend: sends the object to Python, converting it via methods for asServerObject and returns a proxy for the converted object.
- pythonEval: evaluates the expr string subsituting the arguments.
- pythonCommand: evaluates the expr string subsituting the arguments; used for a command that is not an expression.
- pythonCall: call the function in Python, with arguments given.
- pythonGet: converts the proxy object that is its argument to an R object.

- pythonSource: evaluate the file of Python source.
- pythonDefine: define a Python function
- pythonSerialize: serialize the object in Python, via pickle
- pythonUnserialize: unserialize the file in Python, via pickle
- pythonName: return the name by which this proxy object was assigned in Python
- pythonShell: Start an interactive Python shell. See the chapter file in the documentation, section 14.3.

---

ipython *Write a File of Python Commands to Test Package Modules in Python*

---

## Description

A file of python commands will be written that set up an interactive Python session having imported the contents from a file (module) of python code in an R package. Typically, uploading such a file to ipython notebook allows the python code, along with additional or modified code, to be tested directly without interfacing from R.

## Usage

```
ipython(file, package, module = "", ..., RPython = TRUE,
  folder = "python")
```

## Arguments

| | |
|---|---|
| file | A file name or open write connection. The python commands generated will be written to this file. |
| package | The R package containing the relevent module |
| module | The file (module) to be imported. Specifically, a command "from ... import *" will be generated. Omit this argument or supply it as "" to suppress this command, in which case explicit commands should be provided. |
| ... | Additional python commands to be appended to the output file. |
| RPython | Should the path include the XRPython code, default TRUE, which is usually what you want. |
| folder | The name of the folder in the installed package; the default is the suggested "python"; that is, the installed version of folder "inst/python" in the source for the package. Note that it's the installed version; changes to the source code must be installed to show up in the output. |

## isinstance *Test if a Proxy Object is an Instance of a Python Type*

### Description

Applies the Python function `isinstance()` to `object`. NOTE: this function should be used to test inheritance on the Python side, even if there are proxy classes for everything involved. It is not true (with the present version of the package) that inheritance in Python corresponds to inheritance in R for the proxy classes.

### Usage

```
isinstance(object, type, .ev = RPython())
```

### Arguments

| | |
|---|---|
| `object` | Any object. The function returns `FALSE` without further testing if the object is not a proxy object. |
| `type` | A character string corresponding to the Python type (not to the name of a proxy class for the type).<br>A Python error will result if there is no such type, or if `object` is a proxy from another language. The implementation diverges from a direct mapping into the Python `isinstance` to handle a Python bizarre for functions: although `type(f)` causes you to think functions have the obvious type, that doesn't work in `isinstance`. So the R code uses what works for this case. (Before we get too sarcastic, the problem is similar to that in R from primitives, making `class(f)` and `typeof(f)` confusing.) |
| `.ev` | an XRPython evaluator, by default and usually the current evaluator. |

## list_Python-class *Proxy Class for Python Lists*

### Description

This class is a proxy for ordinary list objects in Python. All the standard Python methods for such objects (e.g., `append()`) are available, but methods for R functions such as `]` are not implemented because Python operators do not behave functionally. Instead, additional methods are defined for the proxy lists, e.g., `el(i)`.

### Methods

`append(..., .ev = XRPython::RPython(), .get = NA)` Python Method: append() L.append(object) – append object to end

`count(..., .ev = XRPython::RPython(), .get = NA)` Python Method: count() L.count(value) -> integer – return number of occurrences of value

`el(i, .ev = XRPython::RPython(), .get = NA)` Extract an element from the list (zero based indexing). The index will be coerced to integer (unless a proxy).

`extend(..., .ev = XRPython::RPython(), .get = NA)` Python Method: extend() L.extend(iterable) – extend list by appending elements from the iterable

`index(..., .ev = XRPython::RPython(), .get = NA)` Python Method: index() L.index(value, [start, [stop]]) -> integer – return first index of value. Raises ValueError if the value is not present.

`insert(..., .ev = XRPython::RPython(), .get = NA)` Python Method: insert() L.insert(index, object) – insert object before index

`pop(..., .ev = XRPython::RPython(), .get = NA)` Python Method: pop() L.pop([index]) -> item – remove and return item at index (default last). Raises IndexError if list is empty or index is out of range.

`remove(..., .ev = XRPython::RPython(), .get = NA)` Python Method: remove() L.remove(value) – remove first occurrence of value. Raises ValueError if the value is not present.

`reverse(..., .ev = XRPython::RPython(), .get = NA)` Python Method: reverse() L.reverse() – reverse *IN PLACE*

`sort(..., .ev = XRPython::RPython(), .get = NA)` Python Method: sort() L.sort(cmp=None, key=None, reverse=False) – stable sort *IN PLACE*; cmp(x, y) -> -1, 0, 1

---

## Modules *Import a Python module or add a directory to the Python Search Path*

---

### Description

If called from the source directory of a package during installation, both `pythonImport` and `pythonAddToPath()` also set up a load action for that package. The functional versions, not the methods themselves, should be called from package source files to ensure that the load actions are created.

### Usage

```
pythonImport(..., evaluator, where = topenv(parent.frame()))

pythonAddToPath(directory = "python",
  package = utils::packageName(topenv(parent.frame())), pos = NA, evaluator,
  where = topenv(parent.frame()))
```

### Arguments

| | |
|---|---|
| ..., where | arguments for the $Import() method. See the method documentation for details. |
| evaluator | The evaluator object to use. Supplying this argument suppresses the load action. |
| directory | the directory to add, defaults to "python" |
| package, pos | arguments package and pos to the method, usually omitted. |

### Functions

- pythonImport: Add the module and name information specified to the objects imported for Python evaluators.
- pythonAddToPath: Add the directory specified to the search path for future Python objects.

### Examples

```
## Not run:
## How to search from a local directory, import a function from a file there
## and call the function.
## Including the evaluator argument causes the path change and import to happen
## right now, not in a package being loaded
ev <- RPython()
pythonAddToPath("/Users/me/myPython/", package = "",
                evaluator = ev)
pythonImport("funEx", "foo", evaluator = ev)
pythonCall("foo", 1.1, 1.2)

## End(Not run)
```

---

## noScalar *Send a Non-scalar Version of an Object*

---

### Description

Ensures that an object is interpreted as a vector (array) when sent to the server language. The default strategy is to send length-1 vectors as scalars.

### Usage

```
noScalar(object)
```

### Arguments

| | |
|---|---|
| object | A vector object. Calling with a non-vector is an error. |

### Value

the object, but with the S4 bit turned on. Relies on the convention that XR interfaces leave S4 objects as vectors, not scalars, even when they are of length 1

## References

Chambers, John M. (2016) *Extending R*, Chapman & Hall/CRC. ( Chapter 13, discussing this package, is included in the package: ../doc/Chapter_XR.pdf.)

---

`okPython` *Check for a Valid Python for Interface*

---

## Description

The function returns true or false according to whether a Python interface can be established. This will fail if no Python exists, if it is incompatible with this version of XRPython (e.g., 32 vs 64 bits in Windows), or if for some reason it can't evaluate a trivial expression correctly. Warnings are printed but ignored.

## Usage

```
okPython(verbose = FALSE)
```

## Arguments

| | |
|---|---|
| `verbose` | Should a message with the cause of a failure be reported? Default `FALSE`. |

---

`PythonClassDef-class` *Class and Generator for Python Class Description from Python Metadata*

---

## Description

A description of a Python class, obtained from the Python class definition object. This class extends the `"ServerClassDef"` class in the XR package.

## Fields

`className` the name of the Python class ##

`module` the name of the Python module

## PythonFunction-class *Proxy Objects in R for Python Functions*

### Description

A class and generator function for proxies in R for Python functions.

### Details

An object from this class is an R function that is a proxy for a function in Python. Calls to the R function evaluate a call to the Python function. The arguments in the call are converted to equivalent Python objects; these typically include proxy objects for results previously computed through the XRPython interface.

### Slots

`name` the name of the server language function

`module` the name of the module, if that needs to be imported

`evaluatorClass` the class for the evaluator, by default and usually, `PythonInterface`

`serverDoc` the docstring from Python, if any.

`serverArgs` the Python argument names (not currently used).

## PythonInterface-class *An Interface to Python*

### Description

The `"PythonInterface"` class provides an evaluator for computations in Python, following the structure in the XR package. Proxy functions and classes allow use of the interface with no explicit reference to the evaluator. The function `RPython()` returns an evaluator object.

### Details

The class extends the `"Interface"` class in the XR package and has the same fields. Python-specific methods use the rPython low-level interface. See the Chapter from the "Extending R" book in the documents for this package for details.

### Methods

`Define(text, file)` Define a Python function from a character vector, ‘text‘ or by reading the text from a file via readLines(). Character vectors are taken to represent lines of Python code in a function definiition. The method returns a proxy function with a name inferred from the first line of the text.

`Import(module, ...)` The Python version of this method replaces the general version in XR with the "import" or "from ... import" directives in Python as appropriate. Returns the ‘reticulate‘ version of the module object, which can be used directly.

`initialize(...)` The Python version, with special defaults for prototypeObject and modules

`PythonCommand(strings)` A low-level command execution, needed for initializing. Normally should not be used by applications since it does no error checking; use $Command() instead.

`ServerAddToPath(serverDirectory, serverPos)` The Python version using sys.path.append()

`ServerClassDef(Class, module = "", example = TRUE)` The Python version using PythonClassDef()

`ServerEval(strings, key = "", get = NA)` The Python version using value_for_R()

`ServerFunctionDef(name, module = "")` The Python version using PythonFunction()

`ServerRemove(key)` The Python version using del_for_R())

`ServerSerialize(key, file)` Serializing and unserializing in the Python interface use the pickle structure in Python. Serialization does not rely on the R equivalent object.

`ServerUnserialize(file, all = FALSE)` The Python unserialize using unpickle

`Shell(endCode = "exit", prompt = "Py>: ", cont = "Py+: ")` Starts an interactive Python shell. Each line of input must be a complete Python expression or command, which will be evaluated in the same context as $Eval() expressions. To continue over multiple lines, end all but the last with an unescaped backslash. The argument ‘endCode‘ is the string to type to leave the shell, by default "exit".

`Source(filename)` The $Source() method uses the Python function execfile() and therefore is quite efficient.

---

## PythonObject-class *Proxy Objects in R for Python Objects*

---

### Description

This is a class for all proxy objects from a Python class with an R proxy class definition. Objects will normally be from a subclass of this class, for the specific Python class.

### Details

Proxy objects returned from the Python interface will be promoted to objects from a specific R proxy class for their Python class, if such a class has been defined.

## pythonTask *Register an Evaluator Command or Expression at Initialization*

### Description

An unevalated command or expression for the interface is supplied, typically using `quote()` or `substitute`. When an evaluator from the class is created, this command will be evaluated. Repeated calls to this function, to `serverAddToPath()` and to `serverImport()` will evaluate the corresponding requests, in the order in which the corresponding calls took place (typically in the source of a pacakage).

### Usage

```
pythonTask(command)
```

### Arguments

| | |
|---|---|
| `command` | an *unevaluated* command or expression for the evaluator. |

## RPython *An Evaluator for the Python Interface.*

### Description

Returns an evaluator for the Python interface. Starts one on the first call, or if arguments are provided; providing argument `.makeNew = TRUE` will force a new evaluator. Otherwise, the current evaluator is returned.

### Usage

```
RPython(...)
```

### Arguments

| | |
|---|---|
| `...` | arguments to control whether a new evaluator is started. Normally omitted. |

### Details

See `PythonInterface` for details of the evaluator.

## Examples

```
if(okPython(TRUE)) {
  ev <- RPython()
  xx <- ev$Eval("[1, %s, 5]", pi)
  xx
  xx$append(4.5)
  ev$Command("print %s", xx)
}
```

---

`setPythonClass` *Create a Proxy Class for a Python Class*

---

## Description

An R class is defined to act as a proxy for a specified Python class. This specializes the `setProxyClass` function using Python facilities for finding the class definition.

## Usage

```
setPythonClass(Class, module = "", fields = character(), methods = NULL,
  ServerClass = Class, where = topenv(parent.frame()),
  contains = character(), proxyObjectClass = "PythonObject", ...,
  example = TRUE)
```

## Arguments

| | |
|---|---|
| `Class` | the Python name for the class. |
| `module` | the Python module, if this is not a standard library class. |
| `fields, methods, where, ...` | arguments to `setProxyClass` and usually omitted. |
| `ServerClass, contains, proxyObjectClass` | ditto. |
| `example` | an optional (proxy for) an object from the class, to be used to define the fields in the class. If omitted, the interface tries to create a standard object from the class by calling the Python generator with no argument. Argument `example` can also be supplied as `FALSE` to suppress generating the default object. |

## Details

The methods and (inferred) fields of a Python Class are determined and returned consistently with the XR structure. Python classes are coded as class objects in Python, but only the methods are fixed and defined. Objects from the class can have any fields, usually created at initialization time but entirely legal to be added by other methods later. By default, the initialize method tries to create an object from the class, with no arguments in the call to the class generator. Supply the `example` argument to override.

setupStep *The Setup Step*

## Description

The file "setup.R" in the tools directory is designed to create an explicit definiion of proxy classes for the "list" and "dict" types in Python. The file would be run through XR::packageSetup() when creating or modifying these classes in the XRPython package. It provides a useful example for the general task of creating an explicit, written-out version of a proxy class.

## Details

The setup step generates a file "R/pythonProxyClasses.R" in the source directory for the package. The setup step needs to be run twice, first to generate the R code in that file, and again to use the roxygen2 package to generate documentation.

For the first round, the package needs to be installed with an empty version of the file (the file has to exist because the package uses a Collate: directive that mentions it. Running packageSetup() this time defines the proxy classes and dumps them (with some extra stuff) to the target file. and adds a line to the NAMESPACE to export both classes. (If we were willing to let roxygenize() create the namespace directives this would be automatic, but I'm not willing.)

Now the package needs to be installed again, this time with the proxy classes, The second pass of the setup file runs roxygenize(). Finally, as usual with roxygenize(), the package has to be installed one more time to generate the actual documentation.

# Index