Package 'XR'

May 8, 2026

Type Package

Title A Structure for Interfaces from R

Version 0.7.2

Date 2018-03-01

Author John M. Chambers

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

Description Support for interfaces from R to other languages,
built around a class for evaluators and a combination of functions, classes and methods for communication. Will be used through a specific language interface package. Described in the book ``Extending R".

License GPL (>= 2)

Imports methods, utils, jsonlite

NeedsCompilation no

RoxygenNote 6.0.1

Repository CRAN

Date/Publication 2018-03-18 22:31:39 UTC

Contents

`asJSONS4` *Convert an Object to a Dictionary or Array in JSON Notation*

Description

The general converter for an object with a formal class or an object to be described via its S3 class, data and attributes. Not usually called directly, but from `objectAsJSON` or method in an interface package.

Usage

```
asJSONS4(object, prototype, exclude = character(), level = 0)
```

Arguments

`object`	The object to convert.
`prototype`	The prototype object (supplied from the evaluator).
`exclude`	Slots to exclude from the dictionary.
`level`	The level of expansion of objects within the original object.

References

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

asRObject *Specialize the R Object Returned from an Interface Evaluator*

Description

The result of a server language expression is returned as a string, using the JSON standard notation to represent a scalar, list or dictionary. Methods for this function get the simple R object obtained from deparsing and interpret it generally.

Usage

```
asRObject(object, evaluator)

## S4 method for signature 'ProxyObject'
asRObject(object, evaluator)

## S4 method for signature 'vector_R'
asRObject(object, evaluator)

## S4 method for signature 'list'
asRObject(object, evaluator)

## S4 method for signature 'data.frame'
asRObject(object, evaluator)
```

Arguments

`object`	An object constructed from the explicit representation as a dictionary. The elements of the dictionary will be converted into objects for the slots of the same name. Application-written methods will re-interpret the object into the intended R form, not necessarily from the same class.
`evaluator`	This argument will be supplied as the evaluator object doing the conversion. Therefore, methods may have one of the specific evaluator classes (e.g., `"PythonInterface"`, in their signature.

Details

The methods supplied with the 'XR' pacakage handle the standard mechanisms for interpretation. Additional methods are likely to interpret proxy class objects for which the standard XR representation in terms of class and slots is not what's actually wanted.

Methods (by class)

- `ProxyObject`: When a proxy object appears, usually as an element of a list, it is expanded, by using the 'Get()' method of the evaluator and calling 'asRObject()' on the result.

- `vector_R`: To distinguish typed R vectors from a general JSON list, encode the desired data as an object from the "vector_R" class. Vector types whose elements cannot be represented in JSON (e..g, "complex") should be returned as a list of character strings in a format that R will parse as elements of the suitable vector object. (e.g, ".5+3i")
- `list`: Both lists and dictionaries will come here from the basic conversion. ‘names(object)‘ will either be NULL or all non-empty, from a dictionary.
- `data.frame`: Assume this has been done via .RClass; avoid inheriting the list method

Writing Application Methods

Application packages will typically write methods for special classes, and often for classes themselves defined in the package. One good reason is that the server language does not naturally return the eventually intended object in a convenient form; for example, because it does not have typed arrays. Then a special class will be defined in R. The server code will generate a dictionary with the `".RClass"` element having the class name, plus whatever slots make sense. The application method for `asRObject()` will take these slots and construct whatever object is really intended. For an example, see the method for class `vector_R`.

References

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

`asServerObject` *Generate a Server Language Expression corresponding to an R Object*

Description

Returns a string that can be inserted into a server language expression. When parsed and evaluated by the server evaluator, the result will be the appropriate object or data.

Usage

```
asServerObject(object, prototype)

## S4 method for signature 'name'
asServerObject(object, prototype)

## S4 method for signature 'AssignedProxy'
asServerObject(object, prototype)

## S4 method for signature 'ProxyFunction'
asServerObject(object, prototype)

## S4 method for signature 'ProxyClassObject'
asServerObject(object, prototype)
```

Arguments

object	The R object.
prototype	The proxy for a prototype of the server language object wanted. When called from the 'AsServerObject()' method of an evaluator, this argument is supplied automatically from a class of objects for that evaluator, allowing methods to be defined specialized to the various interface evalutor classes.

Details

Methods for proxy objects and proxy class objects will produce the name under which they were assigned. The default method uses JSON to encode the object as a string and expects the server side interface to have a function objectFromJSON() to decode the string.

Methods (by class)

- name: class "name" is used to pass unquoted strings, in case your interface code did an explicit assign (but that's discouraged).
- AssignedProxy: Proxy objects are just passed as their character string, although a particular interface class could do something different, like refer to a table.
- ProxyFunction: a proxy function just turns into its server language name.
- ProxyClassObject: an object from a proxy class will be replaced by the name of the referenced object

References

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

AssignedProxy-class *Class for Assigned Proxy Objects and Related Mechanisms*

Description

The 'AssignedProxy' class is used by interface packages to return a reference to an assigned server object. The R user can then supply this object anywhere in later interface computations, just as one would use the name of an R object in a function call or other expression.

Details

The virtual class 'ProxyObject' is a superclass, designed to allow other mechanisms for proxy objects (none exists at this time).

Slots

`.Data` The 'AssignedProxy' class is a subclass of 'character'; the actual character string will be generated by the interface and is unique over the session, so long as the 'XR' package stays loaded.

`serverClass,module` The server language class and module for the corresponding object.

`size` The size (usually, length) of the server object, if that makes sense. Can be used to make decisions about handling large objects.

`evaluator` The evaluator object that returned the proxy. Having this as a slot allows interface computations to operate directly on the proxy, without a user-supplied evaluator.

References

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

---

dumpProxyFunction *Write A Proxy Function to a File or Connection*

Description

This function is called by the `$SaveProxyFunction()` method of an evaluator object. It is exported for the convenience of packages inheriting from XR and would not normally be called by a user.

Usage

```
dumpProxyFunction(file, object, objName = object@name, docText)
```

Arguments

`file`	the file or connection for writing the function text.
`object, objName, docText`	arguments supplied by the evaluator method.

---

evaluatorAction *Carry Out an Evaluator Initialization Action*

Description

This function is called from the Startup() method of an evalautor and is not useful to be called directly. It is exported to make it visible from within a subclass of "Interface".

Usage

```
evaluatorAction(action, ev)

## S4 method for signature 'language'
evaluatorAction(action, ev)

## S4 method for signature 'pathEl'
evaluatorAction(action, ev)
```

Arguments

action	the action from the table. Must be an expression or some special class, typically a path element to add to the server path.
ev	the evaluator.

Methods (by class)

- language: a language object, just evaluate it.
- pathEl: a "pathEl" object to add to the server search path.

---

evaluatorActions *Add to Table of Search Paths and Import Commands*

Description

Utilities to add to the table of search paths, import commands and tasks for all evalutors of the specified class. Called only from analogous functions in packages for specific languages.

Usage

```
serverAddToPath(Class, directory,
  package = utils::packageName(topenv(parent.frame())), pos = NA,
  onLoad = NA, where = topenv(parent.frame()))

serverImport(Class, ..., onLoad = nzchar(packageName(where))  &&
                 !environmentIsLocked(where),  where = topenv(parent.frame()))

serverTask(Class, command, onLoad = nzchar(packageName(where)),
  where = topenv(parent.frame()))
```

Arguments

Class	the class of the server-specific evalutor.
directory	the directory to add to the search path table.
package	the name of the server-specific interface package.

pos	where in the list of directories to insert this one. Defaults to the end.
onLoad, where	used to set up a load action; should be omitted if called from a package source
...	arguments to pass to the evaluator's $Import() method
command	an *unevaluated* command or expression for the evaluator.

Details

The server-specific information is added to the table stored by the XR package. All future evaluators for the specified interface class will have these directories in their search path, will import the module information specified and carry out any other tasks supplied.

If a current evaluator for this class exists, it applies all the commands, but *previous* evalutors for this class are not modified.

Commands are evaluated in the order of the calls to these functions. For example, the application package should execute a call to add to the search path before any calls to import modules form the corresponding directory.

Functions

- serverAddToPath: Add the directory to the search path of all evaluators of this class.
- serverImport: An import command with these arguments will be executed for each new evaluator of this interface class, and for the current evaluator if one exists.
- serverTask: 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.

References

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

---

fillNames *Utilities for Server-Language Specific Use*

Description

A utility to fill in blank names in list elements to make it valid for a dictionary.

Usage

```
fillNames(object, noNamesOK = FALSE)
```

Arguments

object	a list object, possibly with empty or duplicate names.
noNamesOK	what to do with a list having no names—leave it alone or fill them all in?

References

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

`fixHelpTopic` *Make a Help Topic an Explicit Character String*

Description

A helper function to pass on a help topic (specifically for a reference-class method) to another help-style function; i.e., so the user could have supplied either a name or a general expression that evaluates to a character string.

Usage

```
fixHelpTopic(topic)
```

Arguments

`topic`	The *expression* supplied by your user. If this was a name, it will be taken literally, otherwise evaluated two levels up the call stack, which should return a character string.

References

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

`from_Server-class` *A Class to Describe General Server Objects*

Description

Classes that inherit from this class are used to convert to R a server language object that is composed of named fields. The corresponding R object will have conversions for the fields that can be accessed with the "$" operator.

Usage

```
## S4 method for signature 'from_Server'
initialize(.Object, ..., referenceClass = TRUE)

## S4 method for signature 'from_Server'
x$name

## S4 method for signature 'from_Server'
show(object)
```

Arguments

.Object, referenceClass	arguments supplied automatically.
...	possible slots for subclasses
x, name	the object and the field name
object	an object from some server class, converted by the $Get() method or an equivalent computation, such as supplying .get = TRUE to a proxy function call.

Methods (by generic)

- initialize: performs some checks on the fields
- $: extract a field. The name must match a field in the data part.
- show: automatic printing, currently just a list of field names.

Slots

serverClass the name of the server language class

module the name of the server language module

language the name of the server language

fields the names of the server language fields

data the converted data for the server fields

---

getInterface *Get or start an evaluator for an interface*

Description

Utility functions to manage a table of evaluators, indexed by the evaluator class, typically one class per server language. All are typically hidden by functions or methods for the particular class. rmInterface and evaluatorNumber are used by methods and exported so that subclasses of interface evaluators will have access to them.

Usage

```
getInterface(Class, ..., .makeNew = NA, .select = NULL)

rmInterface(evaluator)

evaluatorNumber(evaluator, add = length(id) > 0)
```

Arguments

Class	the name of the interface class for this evaluator; by default, the class of the current evaluator. Can also be the class definition object.
...	arguments, if any, are passed to the generator for the evaluator
.makeNew	can be used to force or prevent starting a new evaluator, if passed as a logical value. Can also be passed as a function that tests the suitability of a current evaluator, returning TRUE if this one won't do, and a new one should be generated instead (consistent with the ... arguments, presumably). The default is NA, meaning that an existing evaluator is OK, but one should be generated if none exists. In contrast, FALSE means to return NULL if no matching evaluator exists.
.select	Can be supplied as a function of one argument, which will be called for each evaluator of this class and which should return TRUE/FALSE according to whether the evaluator should be accepted. Allows applications to select, for example, a particular evaluator corresponding to a known connection.
evaluator	any evaluator object.
add	if this evaluator is not in the table, add it. Default TRUE.

Details

Specific language interface packages usually supply a convenience function equivalent to calling `getInterface()` for their class; e.g., `RPython()` in `'XRPython'`

If no `Class` is given, the current (i.e., last active) evaluator is returned

Value

`getInterface()` returns an interface evaluator for this class, starting one if none exists.

Functions

- `rmInterface`: Remove the specified evaluator from the table of available interfaces.
- `evaluatorNumber`: Return the sequential number for this evalutor; used in `ProxyName()` method. If not there: if `add`, add the evaluator to the table; else return `NA`.

Examples

```
## the current evaluator, or NULL if none exists
getInterface()
## this will always be NULL, because no evaluator has this class
getInterface("Interface", .makeNew = FALSE)
```

`Interface-class` *Reference class for all interface evaluators*

Description

This class has the fields required for any specific interface and the methods that are defined centrally in the XR structure. As noted in the documentation for inidividual methods, some methods must be redefined in the specific interface.

Fields

`evaluatorId` A character string (usually unique) giving the language and date when started

`languageName` The server language. Does not have to be unique if multiple classes implement interfaces to the same language.

`proxyCount` Counter used to generate unique names for proxy objects.

`propertyFormat` C-style format string for access to properties and methods in this evaluator. Nearly always just the two names, separated by "."

`proxyClassTable` An environment for all proxy classes known currently for this evaluator class.

`prototypeObject` The object representing any proxy class for this inteface. Usually from a class defined by the specific interface package, to distinguish its proxy classes. This field is passed as the `prototype` argument in calls to `asServerObject`.

`simplify` Should lists whose elements are each basic scalars be unlisted? Default FALSE. May also be a function that takes a possibly simplifiable list as argument and returns the vector/list result. Use this to apply a customized test; e.g., all scalars must have same type.

`propertyFormat` The C-style format for a property (i.e., field) in the server language. The default assumes `"."` is the field operator, as in all likely server languages so far.

`proxyClassTable` Used to keep track of proxy classes encountered

`modules` The evaluator's table of currently imported modules.

`serverPath` The evaluator's current server language path for importing.

Methods

`AddToPath(directory = base::tolower(languageName), package = utils::packageName(topenv(parent.frame()))` Add the directory to the systemPath By default, appends to the path; if 'pos' is given, inserts at that position. If both directory and package are omitted, the method looks for the package name in the calling function (suitable if the method call is from a package source file).

`AsRObject(object)` Given an R object made up of vectors, lists and named tables, interpret that as a general R object, using a convention that may be specialized to the server language by overriding $AsRObject() or by methods for asRObject(). The argument will may be from a proxy class.

`AsServerObject(object, prototype = prototypeObject)` Given an R object return a string that, when evaluated in the server language gives a corresponding object in that language. The default implementation uses the function XR::objectAsJSON, which returns a JSON string and assumes a function 'objectFromJSON(string)' in the server. The conversion may be specialized to server language classes by methods for asServerObject() or objectAsJSON().

`Call(fun, ..., .get = NA)` Call the server language function 'fun'. Each of the '...' arguments will be translated into a server language expression by the AsServerObject() method.

`Command(expr, ...)` Like Eval(), but the value of the expression is ignored. In particular, may be a command in the server language that is not an expression.

`Eval(expr, ..., .get = NA)` Evaluate 'expr' and return the value, possibly as a proxy. Expressions are supplied as character strings to be parsed and evaluated by the server language. If 'expr' has " with the appropriate server language code equivalent to the '...' arguments. If 'expr' has more than one element, all but the last are evaluated by $Command(), with '...' ignored.

`finalize(...)` method called when the object is garbage collected. A call to the $Quit() method also calls this method (recalling it later then does nothing). In case some server action (like closing down a subprocess) is required, the $ServerQuit() method is called, and the evaluator is then removed from the table of interface evaluators.

`Function(serverFun)` Returns an R function object that calls the specified sever language function, specified by its name in the server language or by the proxy object returned by the $Define() method of the evaluator

`Get(what, ...)` Return the value, always converted to an R object. Usually gets a proxy object as the argument, but can be called like $Eval(), if ... is non-empty.

`Import(module, ...)` Import the module. The "Interface" method assumes a command "import" in the server language and does not handle any extra arguments (e.g., for importing specific members).

`initialize(...)` initializes the evaluator in a language-independent sense.

`MethodCall(object, name, ..., .get = NA)` Call the server language method 'name' on 'object', with arguments '...', by default assuming a language in which the syntax is 'object.name(...)'. To override with a different syntax, define field propertyFormat in the evaluator. Note that 'name' must be a character string, not an evaluation in the server.

`MethodEval(string, catch = FALSE, print = FALSE)` The string is a method call for the evaluator, or the name of a field. Evaluated as the expression ev$string.

`New(serverClass, serverModule = "", ...)` Generate a new object from the specified server class. The corresponding generator function in the server is given by ServerGenerator(serverClass), by default just the class name. Typically called from the $Initialize() method of the proxy class.

`ProxyClassName(serverClass)` If there is a proxy class defined corresponding to this serverClass, return the name of that class (typically pasted with the server langauge, separated by underscore). If no such class is defined, return NA.

`ProxyClassObject(object)` If 'object' is an assigned proxy, check whether the serverClass is a known proxy class and if so, return an object from that class; otherwise return 'object'.

`ProxyName(x, new = TRUE)` Called without arguments, returns a key for the next proxy object. In the default strategy, this is a string "R_i_j" where i is the sequence code for the evaluator and j is the proxy count, incremented if 'new' is TRUE. If 'x' is supplied as an existing proxy object, returns the key for that object.

`SaveProxyFunction(save, object, objName = obj@name, docText = NULL)` The object is an expanded function definition, provided by the initialize method for this class. 'save' should be either an environment in which to assign it or a place to dump the R source, either an open connection or a file name.

`Send(object, serverClass = "", .key = NULL)` Send the converted version of ‘object‘ to the server language. If ‘.key‘ is specified, assign it under that name. By default (and recommended) a proxy object in R provides the name for the converted object. If ‘serverClass‘ is supplied, there should be a corresponding asServerObject() method.

`Serialize(object, file, append = FALSE)` Use the server language serialization to serialize ‘object‘ to the specified ‘file‘. According to ‘append‘ either append to the file (default) or overwrite. The supplied object should be a proxy for a server language object.

`ServerClass(Class, module)` If possible, return the class structure of Class, a class in the server language. module= is the server module/package/library in which Class is defined, or "". If no reflection information is available, return NULL (which this definition does). Should return a list or reference object: "$fields" and "$methods" should be character vectors or named lists of the server fields and metohds.

`ServerClassDef(Class, module, ...)` Individual interface packages will define this to return a named list or other object such that value$fields and value$methods are the server fields and methods, character vectors of names or named objects whose elements give further information. This default version returns NULL, indicating that no metadata is available.

`ServerEval(expr, key, get)` Must be defined by the server language interface: evaluates ‘expr‘(a text string). If ‘key‘ is an empty string, ‘expr‘ is treated as a directive, with no defined value. Otherwise, ‘key‘ is a non-empty string, and the server object should be assigned with this name. The value returned is the R result, which may be an AssignedProxy() object. If ‘get‘ is TRUE or the value judged simple enough, it will be converted to an ordinary R object instead.

`ServerExpression(...)` The arguments define an expression in the server language. The first argument is a string; any others are objects to be substituted for in the string. These can include proxy objects or R data.

`ServerFunctionDef(name, module = "", ...)` The XR method defines the proxy function with no special metadata information. Server langugae metadata may be used by a method that overrides this one, and calls it.

`ServerRemove(key)` Should be defined by the server language interface: The reference previously created for ‘key‘ should be removed. What happens has no effect on the client side; the intent is to potentially recover memory.

`ServerSerialize(key, file)` Serialize the proxy function corresponding to ‘key‘ to the specified ‘file‘. Will normally be defined using the serialization supported by the particular server language. The default gets the object and serializes in R, so only works if conversion does

`ServerUnserialize(file, all)` Unserialize the file, returning a proxy object for a list, or equivalent in the server language, of all the objects serialized to this file. Because open connections can not generally be shared among languages, must unserialize the entire file.

`Shell(endCode = "quit", prompt = ">»: ", cont = "+++: ")` Starts an interactive shell. Each line of input must be a complete expression or statement in the server language. To continue over multiple lines, append an unescaped backslash to all but the last line.

A line typed to the shell starting with "$" is an escape back to the evaluator, and can be used to call evaluator methods, e.g., "ProxyName(x)". See the example in the documentation for class "Interface" in XR

`Source(filename)` Parse and evaluate the contents of the file. This method is likely to be overriden for particular langauges with a directive to include the contents of the file. The ‘XR‘ version reads the file and processes the entire contents as a single string, newlines inserted between lines of the file.

`startupActions()` Perform the evaluator actions specified for this object, typically additions to search path and imports

`Unserialize(file, all = FALSE)` Unserialize a list of objects previously written to 'file' by $Serialize(). Returns a list of proxy objects. If all=FALSE, returns a single object if exactly one object found.

References

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

`InterfaceCondition-class` *Classes of objects representing errors or other conditions in a server language*

Description

Errors and warnings generated in evaluating an expression in the server language will be returned to R as objects from one of these classes. The interface evaluator will normally throw an error for `"InterfaceError"` and issue a warning for `"InterfaceWarning"`.

Slots

`message` The character string message from the server language evaluator.

`value` In the case of a warning, the object to return after issuing the condition.

`expr` The expression sent to the server language that produced the condition.

`evaluator` The interface evaluator object receiving the condition.

References

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

`isProxy` *Test if an Object is a Proxy*

Description

Returns `TRUE` if `object` is either a simple proxy or an object from a proxy class.

Usage

```
isProxy(object)
```

Arguments

`object`	Any object.

MiscMethods *Miscellaneous methods*

Description

Convenience methods are provided for operator $ to give more informative error messages if AssignedProxy objects are assumed incorrectly to have a proxy class definition.

Usage

```
## S4 method for signature 'AssignedProxy'
x$name

## S4 replacement method for signature 'AssignedProxy'
x$name <- value
```

Arguments

x, name, value	Arguments to the operator.

References

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

nameQuote *Plain Double Quote for Names*

Description

Utility to surround names of classes, etc with double quotes (not the single quotes of shQuote or the fancy quotes of dQuote)

Usage

```
nameQuote(what)
```

Arguments

what	the input string. Should be a name or something without quotes at least.

Value

the string with quotes. But empty strings stay empty.

References

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

`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.)

`objectAsJSON` *Construct a String in JSON Notation to Represent an R Object*

Description

The XR structure requires a server-language function `objectFromJSON()` which parses an object description in JSON. Methods for generic function `objectAsJSON()` should produce the appropriate string.

Usage

```
objectAsJSON(object, prototype = prototypeObject(), level = 1)

## S4 method for signature 'array'
objectAsJSON(object, prototype = prototypeObject(),
  level = 1)

## S4 method for signature 'environment'
objectAsJSON(object, prototype = prototypeObject(),
  level = 1)

## S4 method for signature 'list'
objectAsJSON(object, prototype = prototypeObject(),
  level = 1)

## S4 method for signature 'envRefClass'
objectAsJSON(object, prototype = prototypeObject(),
  level = 1)

## S4 method for signature 'Interface'
objectAsJSON(object, prototype = prototypeObject(),
  level = 1)

## S4 method for signature 'AssignedProxy'
objectAsJSON(object, prototype = prototypeObject(),
  level = 1)

## S4 method for signature 'ProxyClassObject'
objectAsJSON(object,
  prototype = prototypeObject(), level = 1)
```

Arguments

`object`	The object to be converted.
`prototype`	The prototype server class; see `asServerObject`.
`level`	Will be 1 for top-level call, incremented when recalled for an element. Used to make choices about scalars.

Details

This function is typically called from a method for `asServerObject`. Methods for `objectAsJSON()` in turn often call one or both of two helper functions: `asJSONS4()`, which produces the full description of the R object; and `typeToJSON()`, which produces the code for the basic R types, ignoring all class or attribute information.

Value

A string that will be parsed according to JSON grammar.

Methods (by class)

- `array`: treat matrix and array objects as legitimate S3 objects
- `environment`: An environment is encoded with its class, in contrast to a named list which may be a simple dictionary.
- `list`: A list will be encoded as a JSON list if it has no names, as a JSON dictionary if it has all distinct, non-empty names, or with an explicit representation in all other cases.
- `envRefClass`: An explicit representation that includes the fields.
- `Interface`: An interface object is transmitted via its character string Id, enough to identify the object; the rest is too R-dependent to be useful.
- `AssignedProxy`: Gets the object back from the server, then recalls the generic. It's usually not a good idea to get here, because bringing the proxy back and then converting it again is not foolproof; better to make direct use of the proxy. But if a proxy object is part of an ordinary list, environment or other R object, this method will be used.
- `ProxyClassObject`: Gets the object back from the server, then recalls the generic. See the comments under the `"AssignedProxy"` method.

References

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

`objectDictionary` *Generate the Explicit Dictionary form for an R Object*

Description

The XR interface strategy uses an explicit named list (i.e., dictionary) to describe an R object from a particular class. This function creates the suitable form for such a dictionary, based on the formal class or the contents of an object. Used by some interface packages (e.g., XRJulia) but likely only of information value otherwise, to tell you how to code an object in the server language.

Usage

```
objectDictionary(object, exclude = character())
```

Arguments

`object`	the object to use to infer the representation
`exclude`	slots or the like that should *not* be in the dictionary form.

Value

a named list with the required entries, e.g., `".RClass"`.

References

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

`packageSetup` *Execute a Setup Step for a Package*

Description

The R code in one or more files is evaluated to carry out setup computations that will write some code, documentation or anything else into the source directory of an R package. Designed for package authors who want to use techniques such as the proxy classes in the XR set of interfaces, `compileAttributes()` in Rcpp or inline documentation in roxygen2.

Usage

```
packageSetup(file = "setup.R", dir = ".", needPackage = TRUE)
```

Arguments

`file`	The name of the files to be parsed and evaluated for the setup step, by default, `"setup.R"`. Will look in the the current directory or the inst/tools directory for a file of this name.
`dir`	Optional directory to use as the working directory for the evaluation. By default the current working directory should be the source directory for the package.
`needPackage`	The package for which the setup is intended. Not needed if the working directory (either currently or given by `dir` is the source directory for that package. If supplied can be either the package name or `FALSE` if the setup is not intended for a package.

Details

The computations will be carried out in an environment constructed by`packageSetup()` with the namespace of the package as its parent, and a `.packageName` object to associate it with the package. By default, the setup step looks for the `DESCRIPTION` file in the working directory to find the package name. The package must have been installed in the library path of this R session.

ProxyClass-class *A Class to Describe Classes in the Server Language*

Description

Used by initialize() methods for proxy class objects and therefore exported for the sake of packages using the XR model. Not typically needed by end users.

Fields

`ServerClass` the name of the corresponding server language class.

`language` the name of the server language (locked)

`evaluatorClass` the class for an interface evaluator for this proxy class.

ProxyClassObject-class *A Class for Objects that are Proxies for Specific Server Class Objects*

Description

This class is extended by all specific proxy classes for a particular language. If a proxy object is returned from the server language whose server class matches a defined proxy class, then an object from that class is generated.

Fields

`.proxyObject` the actual proxy reference

`.proxyClass` the description of the server language class (name, module, langauge)

`.ev` the evaluator that produced this proxy object.

Methods

`proxyName()` the character string under which the server language object is assigned. Useful to examine the object in a shell for the server language.

`size()` returns the server-language size of the object, possibly NA

References

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

`proxyEvaluator` *The Evaluator Function Object Referred to from a Proxy Objec*

Description

Any proxy for a server language object contains a reference to the interface evaluator object used to create the proxy object. This function retrieves the evaluator (whether or not there is a proxy class for this object). The function is called from specialized methods for particular server langauge classes, as part of a package using the XR package. End users will not typically need to call it directly; it is exported to simplify life for the extending package.

Usage

```
proxyEvaluator(object)
```

Arguments

`object`	any proxy object

`ProxyFunction-class` *A Class for Proxy Functions*

Description

A class for functions in R that call functions in a server language. The arguments in a call are converted to equivalent server language objects, via `asServerObject()`. These usually include proxy objects in R for results previously computed through the same interface evaluator.

Details

This class is always subclassed for a particular server language. Proxy functions for that language will use a corresponding evaluator to find metadata about the server function.

Slots

`.Data` the function

`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, identifying which server lanaguage is involved.

`serverDoc` documentation for the server language function

`serverArgs` the formal arguments of the server language function, if known

References

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

proxyName *Return the Server Language Name Corresponding to a Proxy Object*

Description

Interface evaluators pass constructed names to the server language evaluator, arranged to be unique within and between evaluators. This function returns the name when given a proxy object.

Usage

```
proxyName(x)

## S4 method for signature 'AssignedProxy'
proxyName(x)

## S4 method for signature 'ProxyClassObject'
proxyName(x)
```

Arguments

x	an object returned from some computation in the server as a proxy for that server object. May be from a proxy class, but doesn't need to be.

Methods (by class)

- `AssignedProxy`: for this class, the name is the object (which extends class "character")
- `ProxyClassObject`: this class has a proxy object as a field.

ProxyObject-class *Class Union to Represent Proxy Objects*

Description

A virtual class to include all classes that can represent proxy objects in the server language. May be extended by the interface for a particular language.

ServerClassDef-class *The Definition of a Server Language Class*

Description

The Definition of a Server Language Class

Fields

`fields,methods` named lists of the server language fields and methods to be exported

`operators` named list of the server class methods that are "operator overloading" of functions.

`readOnly` the names of any fields that should be made read-only in the R class

References

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

serverFields-class *Class Union for Describing Server Language Fields*

Description

May be extended by the interface for a particular language.

setProxyClass *Create a Proxy Class*

Description

Creates a proxy class of a given name and other requirements. Usually infers fields and methods from server language metadata, but can also use explicitly supplied values. Particular interface packages will typically have a specialized version that calls this function.

Usage

```
setProxyClass(Class, module = "", fields = character(), methods = NULL,
  ServerClass = Class, where = topenv(parent.frame()),
  contains = character(), evaluatorClass,
  proxyObjectClass = "ProxyClassObject", language = if (is.null(evaluator))
  "" else evaluator$languageName, readOnly = NULL, ..., save = FALSE,
  objName = Class, docText = NULL)
```

Arguments

`Class`	the name of the class to be used in the proxy, usually just the server language class.
`module`	the name of the server langauge module if it needs to be imported.
`fields, methods`	explicit field and method information if this cannot be found by inspection. Normally omitted.
`ServerClass`	the name of the server language class, normall defaults to `Class`.
`where`	the environment for the class definition. By default, and nearly always, the namespace of the package in which the call to `setProxyClass()` occurs.
`contains`	explicitly needed superclasses if any.
`evaluatorClass`	the evaluator class to identify the evaluator, e.g. `"PythonInterface"` for Python. By default, the current evaluator class.
`proxyObjectClass`	The general class for proxy objects in this interface. Typically obtained automatically from the `prototypObject` field of the evaluator.
`language`	the server language, taken from the evaluator if one is found.
`readOnly`	character vector of any field names that should be marked read-only.
`...`	extra arguments to pass on to `setRefClass()`.
`save`	If the proxy class is being defined in an application package, use this to write to a source file (see Ch. 12 of Extending R) Default `FALSE`, if the proxy class is being assigned in the installation or load of a package.
`objName`	When using the `save=` argument to write R code, use this name in the assignment expression for the generator object. By default, the name of the class.
`docText`	metadata information supplied by the interface for a particular server language.

Details

A proxy class has fields and methods that are created to use the corresponding fields and methods of the server language, through an interface evaluator. This function normally expects information about the class to be returned by the `$ServerClassDef()` method of the evaluator, specialized to the language. It can also be called with explicit lists for the fields and methods. The actual fields and methods will use the interface to access or call the corresponding code in the server language.

Value

a generator object for the R class, along with the side effect of setting the class definition.

References

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

typeToJSON *Convert a Simple Object to JSON String*

Description

Convert a simple object (should have no attributes or formal class) to JSON string. Called from some specific interface packages, usually to put quotes and escapes into a string.

Usage

```
typeToJSON(object, prototype)
```

Arguments

object	the object to convert; only its type will be used to select format
prototype	the prototype object, supplied from the evaluator calling this function

References

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

Unconvertible-class *Unconverted Server Language Objects*

Description

Objects from this class represent server language objects whose conversion was requested but which are judged (by the server side of the interface) to have no equivalent R form. Rather than generating an error, the interface returns an object of this class, which can have convertible attributes. Fields of a convertible object may be unconvertible without preventing conversion of the rest of the parent object.

Slots

serverClass,serverModule The names of the class and module in the server language.

language The language name (for communicating with users), not the interface class name.

attributes A list with names that should be interpreted as properties of the object.

References

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

`valueFromServer` *Convert the String Returned by a Server Language Interface to an R Object.*

Description

This is the conversion mechanism for results returned from a server language interface. By default, JSON is used to decode the string. Otherwise the result of the basic decoding should be provided as argument `object`. Should be called by the implementation of ServerEval for individual interface classes.

Usage

```
valueFromServer(string, key, get, evaluator, object)
```

Arguments

`string`	the string to be passed to JSON.
`key`	the key if a proxy was allowed, otherwise the empty string.
`get`	the logical controling whether a proxy or a converted value was wanted.
`evaluator`	the evaluator object that issued the server language expression.
`object`	If JSON is not used, the call from the server language method should provide the elementary conversion of the result (without using `asRObject()` and the string argument should be omitted. If JSON is used, `object` should be computed by default from JSON.

Value

the R object implied by the server result.

References

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

`vector_R-class` *A class that facilitates returning R vectors via a list in JSON*

Description

Server language code will return a dictionary in which data= is a JSON-style list and type= is the R vector type desired. See the asRObject() method documentation. Objects from this class can also be generated in R, usually supplying just `data` and generating the other slots from that object's properties.

Slots

`data` The actual vector data.

`type` The string for the R type intended. In spite of the slot name, this really the class; for example, "numeric" rather than "double".

`missing` The index of NA's in this vector. Needed because most server languages only treat 'NaN' for doubles and have no mechanism for 'NA' in other types.

References

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

Examples

```
x <- c(1:2,NA,4:5)
vector_R(x)
```

XR *A Structure for Interfaces from R*

Description

The XR package defines classes and functions that will be extended and used by interfaces from R to other languages. The goals are: a uniform approach; simplicity in programming; users of application packages should be essentially unaware that particular functions and objects are in fact proxies for computations in other languages.

Details

The functions in this package will nearly always be used by other packages, providing an interface to a particular language (see packages XRPython and XRJulia for examples). Users will rarely need to call functions in XR and application packages will rarely need to import it directly.

For further details, see the reference [1] (Chapter 13 is included in the documentation for this package).

References

[1] Chambers, John M. *Extending R.* Chapman & Hall, 2016

Index