# widgetInvoke

April 19, 2009

---

`SimpleW-class` *Class "SimpleW": A Class To Represent A Widget Type*

---

### Description

The `SimpleW` is a virtual class used to provide a framework for representing the type of widget used to display and obtain information in the `widgetInvoke` package. There are currently three subclasses, providing for radio buttons (`RadButtonW`), a drop down list (`DropDownW`) or text entry (`TypeInW`)

### Objects from the Class

Objects can be created by calls of the form `new("SimpleW", ...)`.

### Slots

**`defaultValue`:** Object of class `"character"`: This is the only slot defined by `SimpleW` and represents the default value of the argument.

**`nItems`:** Object of class `"numeric"`: This is used by the `RadButtonW` and `DropDownW` subclasses, and describes how many objects are to be represented in their widget.

**`itemNames`:** Object of class `"character"`: This is used by the `RadButtonW` and `DropDownW` subclasses, and is a character vector detailing the names of the objects to display in the widget. The length of `itemNames` must match the value stored in the `nItems` slot.

**`returnType`:** Object of class `"character"`: This is used by the `TypeInW` subclass, and specifies the return type of this argument (e.g. character, numeric, logical).

### Methods

defaultValue `signature(object = "SimpleW")`: Retrieves the defaultValue slot.

nItems `signature(object = "RadButtonW")`: Retrieves the nItems slot.

nItems `signature(object = "DropDownW")`: Retrieves the nItems slot.

returnType `signature(object = "TypeInW")`: Retrieves the returnType slot.

### Author(s)

Jeff Gentry

### See Also

createWF, widgetInvoke

---

createWF *A function to specify widget structure for widgetInvoke*

---

### Description

This function will present a widget (currently using Gtk) that will allow the user (in this case, typically a package author or maintainer) to specify the structure of the function widgets used by widgetInvoke.

### Usage

```
createWF(funName)
```

### Arguments

| | |
|---|---|
| funName | The name of the function to create a widget for |

### Details

The package that funName is in must already be loaded via library for this function to work properly.

This function will use RGtk to display a widget detailing the arguments of the specified function, and other information to be used for the widgetInvoke function. The widget presents a six column table, with the first column detailing each argument name. The arguments themselves are represented by the rows of the table.

The Type column specifies the type of the argument (e.g. character, logical, etc). If this field is blank, it is assumed that any type of value may be entered in by the widgetInvoke user. A character vector can be entered to specify a set of possible values (e.g. "mean", "median") that the argument is allowed to take - to do this use comma separated, quoted strings.

The Default column is used to specify the default value of the argument, if any. Leaving this field blank will imply that there is no default value, otherwise the value in this field must be of the type specified by type.

The Location column describes where this argument will appear. The widget used by widgetInvoke allows for notebook style paning, and this field will specify which pane the argument will appear on. By default, all arguments appear on the "main" pane, but by specifying another string a new pane will be created.

WidgetType allows the user to specify what sort of widget is used by widgetInvoke for the entry of this argument. Currently, logical and vector types must use either Radio or DropDown and all other types must use TypeIn.

If the Required check box is marked for an argument, that means that the argument must be filled in with a value by the widgetInvoke user.

The Reset button can be used to bring the entire table back to its original state. Any changes the user has made will be reverted.

The Check button will check the validity of all the values stored in the table and report to the user if there are potential problems.

The Preview button will display a window that will be identical to what the widgetInvoke user will see. For this sub-window, the Evaluate window will not actually evaluate the function, but simply close the window.

The Save button will save the information in the table to an XML file in the current directory, of the name FUN.xml, where FUN is the name of the function.

## Value

This function is used for its side effect, which is to output a file with the appropriate meta-data. The default filename is funName.xml (where funName is the same as the value specified by the parameter) and stored in the current working directory of the R session. By using the Save As button, this can be changed, and saved to any file the operator wishes.

## Author(s)

Jeff Gentry

## See Also

widgetInvoke

## Examples

```
if (interactive())
  createWF("testWIfun")
```

---

# fun2wFun *A Function To Generate wFun Objects For A Function*

---

## Description

This function will take a function name and attempt to create a corresponding wFun object for it based off of available information about the function.

## Usage

```
fun2wFun(funName)
```

## Arguments

| | |
|---|---|
| funName | The name of the function to use |

## Details

This function will first attempt to get the argument list for the requested function, and then create a basic/default wFun object for this function. The wFun object can be further manipulated by the user, if desired.

Typically, this function is primarily used internally by createWF.

## Value

An object of class wFun, representing available knowledge about the requested function.

## Author(s)

Jeff Gentry

## See Also

createWF, widgetInvoke, wFun, writeWIxml, readWIxml

## Examples

```
z <- fun2wFun("apropos")
funName(z)
funArgList(z)
```

---

# funArg-class *Class "funArg": A Class To Represent A Function Argument*

---

## Description

This class is used by the widgetInvoke to represent the necessary information for creation of widgets for a function argument. Each argument for a function maps to an object of class funArg, and stored as a list with the others in the funArgList slot of the appropriate wFun object.

## Details

The argType slot can specify a particular type, or use the string ANY to allow this argument to be untyped. When using the fun2wFun function, the default for a funArg object is that if there is no argDefault specified by the function, there will be no argType. Likewise, if there is an argDefault, the argType will be of the same type.

## Objects from the Class

Objects can be created by calls of the form new("funArg", ...).

## Slots

**argName:** Object of class "character": The name of the argument

**argDefault:** Object of class "character": The default value of the argument.

**argType:** Object of class "character": The type of the argument (e.g. numeric, character, logical)

**argLocation:** Object of class "character": Which pane of the widget notebook to display this argument.

**argWidgetType:** Object of class "character": What type of widget to use in displaying this argument.

**argRequired:** Object of class "logical": Whether or not this argument is required to have a value for function evaluation.

### Methods

**argDefault** `signature(object = "funArg")`: Retrieves the argDefault slot.

**argLocation** `signature(object = "funArg")`: Retrieves the argLocation slot.

**argName** `signature(object = "funArg")`: Retrieves the argName slot.

**argRequired** `signature(object = "funArg")`: Retrieves the argRequired slot.

**argType** `signature(object = "funArg")`: Retrieves the argType slot.

**argWidgetType** `signature(object = "funArg")`: Retrieves the argWidgetType slot

### Author(s)

Jeff Gentry

### See Also

`createWF`, `widgetInvoke`, `wFun`

### Examples

```
z <- readWIxml(system.file("wFun-example", "apropos.xml",
          package="widgetInvoke"))
a <- funArgList(z)[[1]]
argRequired(a)
argName(a)
argType(a)
```

---

## testWIfun *Find Objects by (Partial) Name*

---

### Description

`testWIfun` returns a character vector giving the names of all objects in the search list matching `what`.

### Usage

```
testWIfun(what, where = FALSE, mode = "any")
```

### Arguments

| | |
|---|---|
| `what` | name of an object, or regular expression to match against |
| `where` | a logical indicating whether positions in the search list should also be returned |
| `mode` | character; if not `"any"`, only objects who's `mode` equals `mode` are searched. |

### Details

If `mode != "any"` only those objects which are of mode `mode` are considered. If `where` is `TRUE`, the positions in the search list are returned as the names attribute.

### Author(s)

Kurt Hornik and Martin Maechler (May 1997).

### See Also

`objects` for listing objects from one place, `help.search` for searching the help system, `search` for the search path.

---

`wFun-class` *Class "wFun": A Class To Represent A Function Widget*

---

### Description

This class is used to model the information used in the `widgetInvoke` function. Each object of this class represents the necessary information for a single function to create its widget interface.

### Objects from the Class

Objects can be created by calls of the form `new("wFun", ...)`.

### Slots

**`funName:`** Object of class `"character"`: The name of the function.

**`funArgList:`** Object of class `"list"`: A list of `funArg` objects, representing the arguments of this function.

### Methods

**funArgList** `signature(object = "wFun")`: Retrieves the `funArgList` slot.

**funName** `signature(object = "wFun")`: Retrieves the `funName` slot.

### Author(s)

Jeff Gentry

### See Also

`createWF`, `widgetInvoke`, `writeWIxml`, `readWIxml`

### Examples

```
z <- readWIxml(system.file("wFun-example", "apropos.xml",
             package="widgetInvoke"))
funName(z)
funArgList(z)
```

`widgetInvoke` *A function to provide a graphical widget to call a function*

## Description

This function will provide a graphical widget (currently using Gtk) to call a function. The widget is defined previously by the package author and information about this widget is stored in an XML file, which is then read in by widgetInvoke. The user's input is passed on directly to the appropriate function.

## Usage

```
widgetInvoke(funName, argOverides=list(), argsOnly=FALSE)
```

## Arguments

| | |
|---|---|
| `funName` | The name of the function |
| `argOverides` | Allows the user to customize the argument display on widgets |
| `argsOnly` | If this is set to `TRUE`, the internal argument list is returned instead of evaluated values. |

## Details

The package that `funName` is in must already be loaded prior to running `widgetInvoke` for `widgetInvoke` to work properly.

Calling this function will display a widget via `RGtk` to the user that allows them to graphically enter in the values for the function described by `funName`. Then, upon hitting the `Evaluate` button, the values stored in the widget are passed directly to the requested function, which is then evaluated normally.

The widget is constructed by information stored in an XML file, which must be generated by running `createWF` for `funName` prior to running `widgetInvoke`.

Some arguments will have default values specified, which can be changed by the user. Also, arguments may have been designated as being required to have a value before evaluation can take place, these arguments will have their name surrounded by `*` characters (e.g. `*colors*` or `*x*`). If the user hits the `Evaluate` button and any of the required parameters do not have a value, the function will not be evaluated and the user will be warned of the situation.

Note that with text entry boxes, values which the user intends to be treated as a character string must be quoted. Any value which is not quoted will be treated in the same manner as usual for R, in that it will be handled as a numerical value, a variable name, etc.

If a user wishes to enter in a vector of values, the simplest manner would be to declare that vector as a variable before calling `widgetInvoke` and then using that variable as the value for an argument. However, if the user wants to enter the vector in the actual entry spot, comma separated values are interpreted to be vectors.

The `argOverides` argument allows the `widgetInvoke` user to customize their widget for a particular function. The structure of this argument is a list, where for any argument that the user wishes to customize, an element is in the list w/ the same name as the argument. That element itself is a list where elements are any slot of the function's `wFun` object's argument list (e.g. `argDefault`). The value of these elements are used to replace the values that come from the XML file. This use of this argument is not recommended for most users, and is primarily intended for the use of other software which is using the `widgetInvoke` function.

### Author(s)

Jeff Gentry

### See Also

createWF, wFun-class

### Examples

```
## Coming soon
```

---

writeWIxml *Functions to read and write widgetInvoke XML files*

---

### Description

These functions are used to serialize object of class wFun to and from XML files.

### Usage

```
writeWIxml(wFun, file = paste(funName(wFun), "xml", sep = "."))
readWIxml(file)
```

### Arguments

| | |
|---|---|
| wFun | An object of class wFun to serialize. |
| file | The filename to use for reading or writing. |

### Details

The class wFun is used to create the user interface in the widgetInvoke function. These wFun objects must be precreated for any given function, and then stored in XML format in the inst/wFun directory of the appropriate package, and are then read in with readWIxml when the user calls widgetInvoke for that function.

The XML format used first defines the primary tag of wFun, which indicates the start of a wFun definition. There are two main sub-fields in the wFun block: funName and funArgList. The former simply contains a string indicating the name of this function. The funArgList block itself contains a series of smaller blocks, each representing an argument to the function.

Within the funArgList are a series of funArg blocks. Within each funArg are six tags. The first is argName, detailing the argument's name. The second is argDefault, which if not empty is the default value of the argument. Next is argType, which describes the type of the argument (e.g. character, numeric, logical, or ANY for an untyped argument). The argLocation field specifies which tab this argument appears on in the widgetInvoke notebook, and argWidgetType describes the type of widget to use in displaying this argument. Finally, the argRequired field is a logical value specifying if this argument is required to be filled in by the user (or a default) before evaluation of the function.

As an example, the following is an example for the function apropos:

```
<?xml version="1.0"?>
<wFun xmlns:bt="http://www.bioconductor.org/WINVOKE">
    <funName>apropos</funName>
    <funArgList>
            <funArg>
                    <argName>what</argName>
                    <argDefault></argDefault>
                    <argType>ANY</argType>
                    <argLocation>main</argLocation>
                    <argWidgetType>TypeIn</argWidgetType>
                    <argRequired>FALSE</argRequired>
            </funArg>
            <funArg>
                    <argName>where</argName>
                    <argDefault>FALSE</argDefault>
                    <argType>logical</argType>
                    <argLocation>main</argLocation>
                    <argWidgetType>Radio</argWidgetType>
                    <argRequired>FALSE</argRequired>
            </funArg>
            <funArg>
                    <argName>mode</argName>
                    <argDefault>"any"</argDefault>
                    <argType>character</argType>
                    <argLocation>main</argLocation>
                    <argWidgetType>TypeIn</argWidgetType>
                    <argRequired>FALSE</argRequired>
            </funArg>
    </funArgList>
</wFun>
```

## Value

The `readWIxml` will return an object of class `wFun` representing the data stored in the specified XML file.

The `writeWIxml` function has no return value.

## Author(s)

Jeff Gentry

## See Also

`createWF`, `widgetInvoke`

## Examples

```
z <- readWIxml(system.file("wFun-example", "apropos.xml",
                package="widgetInvoke"))
writeWIxml(z, file=tempfile())
```

# Index