./doc/langRef.xotcl

Package/File Information

Package provided: XOTcl−langRef 1.3
Package required: Tcl

Defined Objects/Classes:

- *Class*: unknown, alloc, create, info, instdestroy, instfilter, instfilterguard, instforward, instinvar, instmixin, instparametercmd, instproc, new, parameter, parameterclass, recreate, superclass, unknown, volatile.
- *Object*: abstract, append, array, autoname, check, class, cleanup, configure, copy, destroy, eval, exists, extractConfigureArg, filter, filterguard, filtersearch, forward, getExitHandler, hasclass, incr, info, instvar, invar, isclass, ismetaclass, ismixin, isobject, istype, lappend, mixin, move, noinit, parametercmd, proc, procsearch, requireNamespace, set, setExitHandler, trace, unset, uplevel, upvar, vwait.

Filename: ./doc/langRef.xotcl

Description: XOTcl language reference. Describes predefined objects and classes.

Predefined primitives: XOTcl contains three predefined primitives:

`self` computes callstack related information. It can be used in the following ways:

- `self` − returns the name of the object, which is currently in execution. If it is called from outside of a proc, it returns the error message ``Can't find self''.
- `self class` − the self command with a given argument `class` returns the name of the class, which holds the currently executing instproc. Note, that this may be different to the class of the current object. If it is called from a proc it returns an empty string.
- `self proc` − the self command with a given argument `proc` returns the name of the currently executing proc or instproc.
- `self callingclass`: Returns class name of the class that has called the executing method.
- `self callingobject`: Returns object name of the object that has called the executing method.
- `self callingproc`: Returns proc name of the method that has called the executing method.
- `self calledclass`: Returns class name of the class that holds the target proc (in mixins and filters).
- `self calledproc`: Returns method name of the target proc (only applicable in a filter).

- `self next`: Return the "next" method on the path as a string.
- `self filterreg`: In a filter: returns the name of the object/class on which the filter is registered. Returns either 'objName filter filterName' or 'className instfilter filterName'.
- `self callinglevel`: Returns the calling level, from where the actual proc was called from. Intermediary next calls are ignored in this computation. The level is returned in a form it can be used as first argument in `uplevel` or `upvar`.
- `self activelevel`: Returns the level, from where the actual proc was invoked from. This might be the calling level or a next call, whatsever is higher in the stack. The level is returned in a form it can be used as first argument in `uplevel` or `upvar`.

`my someMethod` is a short form for `[self] someMethod` and can only be called in a context of an instproc or an method specific proc. It allows certain optimizations and shorter to write.

`next` executes the "next" method on the precedence order and return with the result.

Date: $Date: 2004/08/17 10:12:55 $

Class: *Class*

Class: Class
Heritage: Object
Procs/Instprocs: unknown, alloc, create, info, instdestroy, instfilter, instfilterguard, instforward, instinvar, instmixin, instparametercmd, instproc, new, parameter, parameterclass, recreate, superclass, unknown, volatile.

Description: This meta−class holds the pre−defined methods available for all XOTcl classes.

Instprocs

- **alloc** *obj ?args?*

  Arguments: **obj**: new obj/class name

  ?args?: arguments passed to the new class after creation

  Description: Allocate memory for a new XOTcl object or class. `create` uses `alloc` to allocate memory. But `create` also calls init and evaluates '−' arguments as method calls. In seldom cases the programmer may want to suppress the `create` mechanism and just allocate memory. Then `alloc` can be used.

  Return: new class name
- **create** *objName ?args?*

  Arguments: **objName**: name of a new class or object

  ?args?: arguments passed to the constructor

  Description: Create user−defined classes or objects. If the class is a meta−class, a class is created, otherwise an object. Create firstly calls `alloc` in order to allocate memory for the new object. Then default values for parameters are searched on superclasses (an set if

found). Then `args` is searched for args starting with '−' followed by an alpha character. These arguments are called as methods. '−' followed by a numerical is interpreted as a negative number (and not as a method). If a value of a method called this way starts with a "a", the call can be placed safely into a list (e.g. "Class c [−strangearg −a−] −simplearg 2"). Finally the constructor `init` is called on the object with all arguments up to the first '−' arg.

The `create` method is called implicitly through the `unknown` mechanism when a class (meta−class) is called with an unknown method. E.g. the following two commands are equivalent

```
Car herby −color red
          Car create herby −color red
```

When a users may want to call the constructor `init` before other '−' methods, one can specify '−init' explicitly in the left to right order of the '−' method. Init is called always only once. e.g.:

```
Class Car −init −superclass Vehicle
```

Return: name of the created instance (result of alloc)

- **info** *args*

Arguments: **args**: info options

Description: Introspection of classes. All options available for objects (see info object) is also available for classes. The following options can be specified:

- `ClassName info classchildren ?pattern?`: Returns the list of nested classes with fully qualified names if `pattern` was not specified, otherwise it returns all class children where the class name matches the pattern.
- `ClassName info classparent`: Returns the class ClassName is nesting to.
- `ClassName info instdefault method arg var`: Returns 1 if the argument `arg` of the instance method `method` has a default value, otherwise 0. If it exists the default value is stored in `var`.
- `ClassName info instfilter`: Returns the list of registered filters. With −guard modifier all instfilterguards are integrated (`ClassName info instfilter −guards`).
- `objName info instfilterguard name`: Returns the guards for instfilter identified by name.
- `ClassName info heritage ?pattern?`: Returns a list of all classes in the precedence order of the class hierarchy. If pattern is specified, only matching values are returned.
- `ClassName info instances ?pattern?`: Returns a list of the instances of the class. If pattern is specified, only matching values are returned.
- `ClassName info instargs method`: Returns the arguments of the specified method.
- `ClassName info instbody method`: Returns the body of the specified method.
- `ClassName info instcommands ?pattern?`: Returns all commands defined for the class. If pattern is specified it returns all commands that match the pattern.
- `ClassName info instinvar`: Returns class invariants.

- `ClassName info instmixin methodName`: returns list of non−positional args of methodName
- `ClassName info instmixin`: Returns the list of instmixins of this class.
- `ClassName info instpost methodName`: Returns post assertions of methodName.
- `ClassName info instpre methodName`: Returns pre assertions of methodName.
- `ClassName info instprocs ?pattern?`: Returns all instprocs defined for the class. If pattern is specified it returns all instprocs that match the pattern.
- `ClassName info parameter`: Returns parameter list.
- `ClassName info subclass ?subclassname?`: Returns a list of all subclasses of the class, if subclassname was not specified, otherwise it returns 1 if subclassname is a subclass and 0 if not.
- `ClassName info superclass ?superclassname?`: Returns a list of all super−classes of the class, if superclassname was not specified, otherwise it returns 1 if superclassname is a superclass and 0 if not.

Return: Value of introspected option as a string.

- **instdestroy** *obj ?args?*

Arguments: **obj**: obj/class name

?args?: arguments passed to the destructor

Description: Standard destructor. Destroys XOTcl object physically from the memory. Can be overloaded for customized destruction process.

In XOTcl objects are not directly destroyed, when a destroy is encountered in a method. Beforehand, the interpreter looks up whether the object is still referenced on the method callstack or not. If not, the object is directly destroyed. Otherwise every occurrence of the object on the callstack is marked as destroyed. During popping of the callstack, for each object marked as destroyed, the reference count is decremented by one. When no more references to the object are on the callstack the object is physically destroyed. This way we can assure that objects are not accessed with [self] in running methods after they are physically destroyed.

Return: empty string

- **instfilter** *args*

Arguments: **args**: instfilter specification

Description: If `$args` is one argument, it specifies a list of instfilters to be set. Every filter must be an XOTcl proc/instproc within the object scope. If `$args` it has more argument, the first one specifies the action. Possible values are `set`, `get`, `add` or `delete`, it modifies the current settings as indicated. For more details, check the tutorial.

Return: if `$args` return empty current instfilters, otherwise empty

- **instfilterguard** *filtername guard*

Arguments: **filtername**: filter name of a registered filter

guard: set of conditions to execute the filter

Description: Add conditions to guard a filter registration point. The filter is only executed, if the guards are true. Otherwise we ignore the filter. If no guards are given, we always execute the filter.

Return: empty string

- **instforward** *name ?options? ?callee? ?args?*

Arguments: **name**: name of forwarder method

?options?: −objscope, −methodprefix string, −default names

?callee?: named of the called command or object

?args?: arguments

Description: Register a method (similar to an instproc) for forwarding calls to a callee (target tcl command, other object). If the forwarder method is called, the actual arguments of the invocation are appended to the specified arguments. In callee an arguments certain substituions can take place:

- %proc: subsituted by name of the forwarder method
- %self: subsituted by name of the object
- %1: subsituted by first argument of the invocation
- %%: a single percent.
- %*tcl−command*: command to be executed; substituted by result.

Additionally each argument can be prefixed by the positional prefix `%@POS` (note the delimiting space at the end) that can be used to specify an explicit position. POS can be a positive or negative integer or the word `end`. The positional arguments are evaluated from left to right and should be used in ascending order. valid Options are:

- `−objscope` causes the target to be evaluated in the scope of the object,
- `−methodprefix string` inserts the specified prefix in front of the second argument of the invocation,
- `−default` is used for default method names.

See tutorial for detailed examples.

Return: empty

- **instinvar** *invariantList*

Arguments: **invariantList**: Body of invariants for the class

Description: Specify invariants for the class. These are inherited by sub−classes. The invariants must hold for all instances. All assertions are a list of ordinary Tcl conditions.

Return: empty string

- **instmixin** *args*

Arguments: **args**: instmixin specification

Description: If `$args` is one argument, it specifies a list of instmixins to be set. Every instmixin must be a defined class. If `$args` has more argument, the first one specifies the action. Possible values are `set`, `get`, `add` or `delete`, it modifies the current settings as indicated. For more details, check the tutorial.

Return: if `$args` empty return current instmixins, otherwise empty

- **instparametercmd** *name*

Arguments: **name**: variable to be provided with getter/setter method

Description: Add a getter/setter command for an instance variable with the specified name. This method is used for example by the parameter method. Example:

```
Class C
C instparametercmd x
C c1 −x 100
puts [c1 x]
```

Return: empty string

- **instproc** *name ?non−pos−args?" args body ?preAssertion? ?postAssertion?*

Arguments: **name**: instance method name

?non−pos−args?": optinal non−positional arguments

args: instance method arguments

body: instance method body

?preAssertion?: optional assertions that must hold before the proc executes

?postAssertion?: optional assertions that must hold after the proc executes

Description: Specify an instance method in the same style as Tcl specifies procs. Optionally assertions may be given. The number of args is either 3 or 5. Therefore, to specify only post−assertions an empty pre−assertion list must be given. All assertions are a list of ordinary Tcl conditions.

Return: empty string

- **new** *?−childof obj? ?args?*

Arguments: **?−childof obj? ?args?**: args passed to create

Description: Convenience method to create an autonamed object. E.g.:

```
Http new
```

creates ::xotcl::__#0, a subsequent call creates ::xotcl::__#1, ...
If `−childof obj` is specified, the new object is created as a child of the specified object.

Return: new object name

- **parameter** *parameterList*

Arguments: **parameterList**: list of parameter definitions

Description: Specify parameters automatically created for each instance. Parameters denote instance variables which are available on each class instance and that have a getter/setter method with their own name. Parameters are specified in a parameter list of the form {p1 p2 ... pn}. p1 ... pn may either be parameter names or definitions of the form {parameterName defaultValue}. If a default value is given, that parameter is created during creation process of the instance object, otherwise only the getter/setter method is created (and the parameter does not exist). The getter/setter method has the same name as the parameter. It gets and returns the parameter, if no argument is specified. With one argument, the parameter is set to the argument value. Example:

```
Class Car −parameter {{doors 4} color}
Car herby −doors 2 −color green
```

Return: empty string

- **parameterclass** *class*

Arguments: **class**: parameter class name

Description: Set the parameter class. The parameter class specifies how parameters are stored and maintained internally. Per default, a method "default" is called, to set the parameter with a default value. I.e.,

```
Class Car −parameter {
  {doors 4}
}
```

is a short form for

```
Class Car −parameter {
  {doors −default 4}
```

```
}
```

For specialized parameter classes other methods can be called, e.g.

```
{doors -default 3 -updateWidget car}
```

Return: empty string

- **recreate** *obj ?args?*

  Arguments: **obj**: obj to be recreated

  ?args?: arbitrary arguments

  Description: Hook called upon recreation of an object. Performs standard object initialization, per default. May be overloaded/−written. It calls another method cleanup which handles actual cleanup of the object during next. That means, if you overload recreate, in the pre−part the object still contains its old state, after next it is cleaned up.

  Return: obj name

- **superclass** *classList*

  Arguments: **classList**: list of classes

  Description: Specify super−classes for a class. "superclass" changes the list of superclasses dynamically to `classList`.

  Return: empty string

- **unknown** *?args?*

  Arguments: **?args?**: arbitrary arguments

  Description: Standard unknown mechanism. This mechanism is always triggered when XOTcl does not know a method called on an object. Supposed that there is no method with the called name, XOTcl looks up the method "unknown" (which is found on the Class Object) and executes it. The standard unknown−mechanism of XOTcl calls create with all arguments stepping one step to the right; in the general case:

  ```
  ClassName create ClassName ?args?
  ```

  Unknown can be overloaded in user−defined subclasses of class.

  Return: Standard unknown mechanism returns result of create

- **volatile**

  Arguments: :

  Description: This method is used to specify that the object should be deleted automatically, when the current tcl−proc/object−proc/instproc is left. Example `set x [Object new -volatile]`

  Return: empty string

Procs

- **__unknown** *name*

  Arguments: **name**: name of class to be created

  Description: This method is called, whenever XOTcl references a class, which is not defined yet. In the following example: `Class C -superclass D` D is not defined. Therefore `Class __unknown D` is called. This callback can be used to perform auto−loading of classes. After this call, XOTcl tries again to resolve D. If it succeeds, XOTcl will

continue; otherwise, an error is generated.

Return: empty string

Class: *Object*

Class: Class
Procs/Instprocs: abstract, append, array, autoname, check, class, cleanup, configure, copy, destroy, eval, exists, extractConfigureArg, filter, filterguard, filtersearch, forward, getExitHandler, hasclass, incr, info, instvar, invar, isclass, ismetaclass, ismixin, isobject, istype, lappend, mixin, move, noinit, parametercmd, proc, procsearch, requireNamespace, set, setExitHandler, trace, unset, uplevel, upvar, vwait.

Description: This class holds the pre−defined methods available for all XOTcl objects. All these methods are also available on classes.

Instprocs

- **abstract** *methtype methname arglist*

  Arguments: **methtype**: instproc or proc

  methname: name of abstract method

  arglist: arguments

  Description: Specify an abstract method for class/object with arguments. An abstract method specifies an interface and returns an error, if it is invoked directly. Sub−classes or mixins have to override it.

  Return: error
- **append** *varName args*

  Arguments: **varName**: name of variable

  args: arguments to append

  Description: Append all of the value arguments to the current value of variable varName. Wrapper to the same named Tcl command (see documentation of Tcl command with the same name for details).

  Return: empty string
- **array** *opt array ?args?*

  Arguments: **opt**: array option

  array: array name

  ?args?: args of the option

  Description: This method performs one of several operations on the variable given by arrayName. It is a wrapper to the same named Tcl command (see documentation of Tcl command with the same name for details).

  Return: diverse results
- **autoname** *?|? name*

  Arguments: **?|?**: Optional modifiers:
  '−instance' makes the autoname start with a small letter.
  '−reset' resets the autoname index to 0.

  name: base name of the autoname

  Description: autoname creates an automatically assigned name. It is constructed from the base name

plus an index, that is incremented for each usage. E.g.:

```
$obj autoname a
```

produces a0, a1, a2, ... Autonames may have format strings as in the Tcl 'format' command. E.g.:

```
$obj autoname a%06d
```

produces a000000, a000001, a000002, ...

Return: newly constructed autoname value

- **check** *options*

Arguments: **options**: none, one or more of: (?all? ?pre? ?post? ?invar? ?instinvar?)

Description: Turn on/off assertion checking. Options argument is the list of assertions, that should be checked on the object automatically. Per default assertion checking is turned off. Examples:

```
o check {}; # turn off assertion checking on object o
o check all; # turn on all assertion checks on object o
o check {pre post}; # only check pre/post assertions
```

info check introspects check options.

Return: empty string

- **class** *newClass*

Arguments: **newClass**: new class name

Description: Changes the class of an object dynamically to `newClass`.

Return: empty string

- **cleanup** *?args?*

Arguments: **?args?**: Arbitrary arguments passed to cleanup

Description: Resets an object or class into an initial state, as after construction. Called during recreation process by the method 'recreate'

Return: empty string

- **configure** *?args?*

Arguments: **?args?**: '−' method calls

Description: Calls the '−' methods. I.e. evaluates arguments and calls everything starting with '−' (and not having a digit a second char) as a method. Every list element until the next '−' is interpreted as a method argument. configure is called before the constructor during initialization and recreation. E.g.

```
Object o -set x 4
```

here:

```
o configure -set x 4
```

is executed.

Return: number of the skipped first arguments

- **copy** *newName*

Arguments: **newName**: destination of copy operation

Description: Perform a deep copy of the object/class (with all information, like class, parameter, filter, ...) to "newName".

Return: empty string

- **destroy** *?args?*

Arguments: **?args?**: Arbitrary arguments passed to the destructor

Description: Standard destructor. Can be overloaded for customized destruction process. Actual destruction is done by instdestroy. "destroy" in principal does:

```
Object instproc destroy args {
  [my info class] instdestroy [self]
}
```

Return: empty string

- **eval** *args*

Arguments: **args**: cmds to eval

Description: Eval args in the scope of the object. That is local variables are directly accessible as Tcl vars.

Return: result of cmds evaled

- **extractConfigureArg** *al name ?cutTheArg?*

Arguments: **al**: Argument List Name

name: Name of the Configure Argument to be extracted (should start with '−')

?cutTheArg?: if cutTheArg not 0, it cut from upvar argsList, default is 0

Description: Check an argument list separated with '−' args, as for instance configure arguments, and extract the argument's values. Optionally, cut the whole argument.

Return: value list of the argument

- **exists** *var*

Arguments: **var**: variable name

Description: Check for existence of the named instance variable on the object.

Return: 1 if variable exists, 0 if not

- **filter** *args*

Arguments: **args**: filter specification

Description: If `$args` is one argument, it specifies a list of filters to be set. Every filter must be an XOTcl proc/instproc within the object scope. If `$args` it has more argument, the first one specifies the action. Possible values are `set`, `get`, `add` or `delete`, it modifies the current settings as indicated. For more details, check the tutorial.

Return: if `$args` return empty current filters, otherwise empty

- **filterguard** *filtername guard*

Arguments: **filtername**: filter name of a registered filter

guard: set of conditions to execute the filter

Description: Add conditions to guard a filter registration point. The filter is only executed, if the guards are true. Otherwise we ignore the filter. If no guards are given, we always execute the filter.

Return: an empty string

- **filtersearch** *methodName*

Arguments: **methodName**: filter method name

Description: Search a full qualified method name that is currently registered as a filter. Return a list of the proc qualifier format: 'objName|classname proc|instproc methodName'.

Return: full qualified name, if filter is found, otherwise an empty string

- **forward** *name ?options? ?callee? ?args?*

Arguments: **name**: name of forwarder method

?options?: −objscope, −methodprefix string, −default names

?callee?: named of the called command or object

?args?: arguments

Description: Register a method (similar to a proc) for forwarding calls to a callee (target tcl command, other object). If the forwarder method is called, the actual arguments of the invocation are appended to the specified arguments. In callee an arguments certain substituions can take place:

- %proc: subsituted by name of the forwarder method
- %self: subsituted by name of the object
- %1: subsituted by first argument of the invocation
- %%: a single percent.
- %*tcl−command*: command to be executed; substituted by result.

Additionally each argument can be prefixed by the positional prefix `%@POS` (note the delimiting space at the end) that can be used to specify an explicit position. POS can be a positive or negative integer or the word `end`. The positional arguments are evaluated from left to right and should be used in ascending order. valid Options are:

- `−objscope` causes the target to be evaluated in the scope of the object,
- `−methodprefix string` inserts the specified prefix in front of the second argument of the invocation,
- `−default` is used for default method names.

See tutorial for detailed examples.

Return: empty

- **hasclass** *?className?*

Arguments: **?className?**: name of a class to be tested

Description: Test whether the argument is either a mixin or instmixin of the object or if it is on the class hierarchy of the object. This method combines the functionalities of istype and ismixin.

Return: 1 or 0

- **incr** *varName ?increment?*

Arguments: **varName**: variable name

?increment?: value to increment

Description: Increments the value stored in the variable whose name is varName. The new value is stored as a decimal string in variable varName and also returned as result. Wrapper to the same named Tcl command (see documentation of Tcl command with the same name for details).

Return: new value of varName

- **info** *args*

Arguments: **args**: info options

Description: Introspection of objects. The following options can be specified:

- `objName info args method`: Returns the arguments of the specified method.
- `objName info body method`: Returns the body of the specified method.

- `objName info class ?classname?`: Returns the name of the class of the current object, if classname was not specified, otherwise it returns 1 if classname matches the object's class and 0 if not.
- `objName info children ?pattern?`: Returns the list of aggregated objects with fully qualified names if `pattern` was not specified, otherwise it returns all children where the object name matches the pattern.
- `objName info commands ?pattern`: Returns all commands defined for the object if `pattern` was not specified, otherwise it returns all commands that match the pattern.
- `objName info default method arg var`: Returns 1 if the argument `arg` of the method `method` has a default value, otherwise 0. If it exists the default value is stored in `var`.
- `objName info filter`: Returns a list of filters. With −guard modifier all filterguards are integrated (`objName info filter −guards`). With `−order` modifier the order of filters (whole hierarchy) is printed.
- `objName info filterguard name`: Returns the guards for filter identified by name.
- `objName info hasNamespace`: From XOTcl version 0.9 on, namespaces of objects are allocated on demand. hasNamespace returns 1, if the object currently has a namespace, otherwise 0. The method `requireNamespace` can be used to ensure that the object has a namespace.
- `objName info info`: Returns a list of all available info options on the object.
- `objName info invar`: Returns object invariants.
- `objName info nonposargs methodName`: Returns non−positional arg list of methodName
- `objName info metadata ?pattern?`: Returns available metadata options.
- `objName info methods`: Returns the list of all method currently reachable for objName. Includes procs, instprocs, cmds, instcommands on object, class hierarchy and mixins. Modifier `−noprocs` only returns instcommands, `−nocmds` only returns procs. Modifier `−nomixins` excludes search on mixins.
- `objName info mixin`: Returns the list of mixins of the object. With `−order` modifier the order of mixins (whole hierarchy) is printed.
- `objName info parent`: Returns parent object name (or "::" for no parent), in fully qualified form.
- `objName info post methodName`: Returns post assertions of methodName.
- `objName info pre methodName`: Returns pre assertions of methodName.
- `objName info procs ?pattern?`: Returns all procs defined for the object if `pattern` was not specified, otherwise it returns all procs that match the pattern.
- `objName info vars ?pattern?`: Returns all variables defined for the object if `pattern` was not specified, otherwise it returns all variables that match the pattern.

Return: Value of introspected option as a string.

- **instvar** *v1 ?v2...vn?*

  Arguments: **v1**: instvar variable

?v2...vn?: optional other instvar variables

Description: Binds an variable of the object to the current method's scope. Example:

```
kitchen proc enter {name} {
  my instvar persons
  set persons($name) [clock seconds]
}
```

Now persons can be accessed as a local variable of the method.
A special syntax is: `{varName aliasName}` . This gives the variable with the name `varName` the alias `aliasName`. This way the variables can be linked to the methods scope, even if a variable with that name already exists in the scope.

Return: empty string

- **invar** *invariantList*

  Arguments: **invariantList**: Body of invariants for the object

  Description: Specify invariants for the objects. All assertions are a list of ordinary Tcl conditions.

  Return: empty string

- **isclass** *?className?*

  Arguments: **?className?**: name of a class to be tested

  Description: Test whether the argument (or the Object, if no argument is specified) is an existing class or not.

  Return: 1 or 0

- **ismetaclass** *?metaClassName?*

  Arguments: **?metaClassName?**: name of a metaclass to be tested

  Description: Test whether the argument (or the Object, if no argument is specified) is an existing metaclass or not.

  Return: 1 or 0

- **ismixin** *?className?*

  Arguments: **?className?**: name of a class to be tested

  Description: Test whether the argument is a mixin or instmixin of the object.

  Return: 1 or 0

- **isobject** *objName*

  Arguments: **objName**: string that should be tested, whether it is a name of an object or not

  Description: Test whether the argument is an existing object or not. Every XOTcl object has the capability to check the object system.

  Return: 1 or 0

- **istype** *className*

  Arguments: **className**: type name

  Description: Test whether the argument is a type of the object. I.e., 1 is returned if className is either the class of the object or one of its superclasses.

  Return: 1 or 0

- **lappend** *varName args*

  Arguments: **varName**: name of variable

  args: elements to append

  Description: Append all the specified arguments to the list specified by varName as separated

elements (typically separated by blanks). If varName doesn't exist, it creates a list with the specified values (see documentation of Tcl command with the same name for details).

Return: empty string

- **mixin** *args*

Arguments: **args**: mixin specification

Description: If `$args` is one argument, it specifies a list of mixins to be set. Every mixin must be a defined class. If `$args` has more argument, the first one specifies the action. Possible values are `set`, `get`, `add` or `delete`, it modifies the current settings as indicated. For more details, check the tutorial.

Return: if `$args` empty return current mixins, otherwise empty

- **move** *newName*

Arguments: **newName**: destination of move operation

Description: Perform a deep move of the object/class (with all information, like class, parameter, filter, ...) to "newName".

Return: empty string

- **parametercmd** *name*

Arguments: **name**: variable to be provided with getter/setter method

Description: Add a getter/setter for an instance variable with the specified name as a command for the obj. Example:

```
Object o
o parametercmd x
o x 100
puts [o x]
```

Return: empty string

- **noinit**

Description: flag that constructor (method `init`) should not be called. Example:

```
Class C
C instproc init {} {puts hu}
C c1 −noinit
```

The object `c1` will be created without calling the constructor. This can be used to draw a snapshot of an existing object (using the serializer) and to recreate it in some other context in its last state.

Return: empty string

- **proc** *name ?non−pos−args?" args body ?preAssertion? ?postAssertion?*

Arguments: **name**: method name

?non−pos−args?": optinal non−positional arguments

args: method arguments

body: method body

?preAssertion?: optional assertions that must hold before the proc executes

?postAssertion?: optional assertions that must hold after the proc executes

Description: Specify a method in the same style as Tcl specifies procs. Optionally assertions may be given. The number of args is either 3 or 5. Therefore, to specify only post−assertions an empty pre−assertion list must be given. All assertions are a list of ordinary Tcl conditions.

Return: empty string

- **procsearch** *procname*

  Arguments: **procname**: simple proc name

  Description: Search for a proc or instproc on an object and return the fully qualified name of the method as a list in proc qualifier format: 'objName|classname proc|instproc methodName'. E.g.,

  ```
  o procsearch set
  ```

  returns

  ```
  ::xotcl::Object instproc set
  ```

  .

  Return: fully qualified name of the searched method or empty string if not found

- **requireNamespace**

  Description: The method `requireNamespace` can be used to ensure that the object has a namespace. Namespaces are created automatically by XOTcl, when e.g. an object has child objects (aggregated objects) or procs. The namespace will be used to keep instance variables, procs and child objects. To check, whether an object currently has a namespace, `info hasNamespace` can be used. Hint: In versions prior to XOTcl 0.9 all XOTcl objects had their own namespaces; it was made on demand to save memory when e.g. huge numbers of objects are created. `requireNamespace` is often needed when e.g. using Tk widgets when variables are to be referenced via the namespace (with `... −variable [self]::varname ...`).

  Return: empty string

- **set** *varname ?value?*

  Arguments: **varname**: name of the instance variable

  ?value?: optional new value

  Description: Set an instance variable in the same style as Tcl sets a variable. With one argument, we retrieve the current value, with two arguments, we set the instance variable to the new value.

  Return: Value of the instance variable

- **trace** *varName*

  Arguments: **varName**: name of variable

  Description: Trace an object variable (see documentation of Tcl command with the same name for details).

  Return: empty string

- **unset** *v1 ?v2...vn?*

  Arguments: **v1**: Variable to unset

  ?v2...vn?: Optional more vars to unset

  Description: The unset operation deletes one or optionally a set of variables from an object.

  Return: empty string

- **uplevel** *?level? command ?args?*

  Arguments: **?level?**: Level

  command ?args?: command and arguments to be called

  Description: When this method is used without the optional level, it is a short form of the Tcl command

`upevel [self callinglevel] command ?args?`. When it is called with the level, it is compatible with the original tcl command.

Return: result of the command

- **upvar** *?level? othervar localvar ?othervar localvar?*

  Arguments: **?level?**: Level

  othervar localvar: referenced variable and variale in the local scope

  ?othervar localvar?: optional pairs of referenced and local variable names

  Description: When this method is used without the optional level, it is a short form of the Tcl command
  `upvar [self callinglevel] othervar localvar ?...?`. When it is called with the level, it is compatible with the original tcl command.

  Return: result of the command

- **vwait** *varName*

  Arguments: **varName**: name of variable

  Description: Enter event loop until the specified variable is set (see documentation of Tcl command with the same name for details).

  Return: empty string

Procs

- **getExitHandler**

  Description: Retrieve the current exit handler procedure body as a string.

  Return: exit handler proc body

- **setExitHandler** *body*

  Arguments: **body**: procedure body

  Description: Set body for the exit handler procedure. The exit handler is executed when XOTcl is existed or aborted. Can be used to call cleanups that are not associated with objects (otherwise use destructor). On exit the object destructors are called after the user−defined exit−handler.

  Return: exit handler proc body

Back to index page.