./doc/langRef.xotcl

Package/File Information

No package provided/required

Defined Objects/Classes:

- *::xotcl::Slot*:
- *Attribute*:
- *Class*: __unknown, allinstances, alloc, create, info, instdestroy, instfilter, instfilterguard, instforward, instinvar, instmixin, instparametercmd, instproc, new, parameter, parameterclass, recreate, superclass, unknown.
- *Object*: abstract, append, array, autoname, check, class, cleanup, configure, contains, 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, subst, trace, unset, uplevel, upvar, volatile, vwait.

Filename: ./doc/langRef.xotcl

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

Predefined primitives: XOTcl contains the following predefined primitives (Tcl commands):

`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 isnextcall`: Return 1 if this method was invoked via next, otherwise 0
- ◊ `self next`: Return the "next" method on the precedence 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, whatever is higher in the stack. The level is returned in a form it can be used as first argument in `uplevel` or `upvar`.

`my methodName`
: is a short form for `[self] methodName` 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`
: invokes the next shadowed (same-named) method on the precedence path and returns its result. If `next` is called without arguments, the arguments of the current method are passed through the called method. If `next` is invoked with the flag `--noArgs`, the shadowed method is called without arguments. If other arguments are specified for next, these will be used for the call.

`myvar varName`
: returns the fully qualified variable name of the specified variable.

`myproc methodName ?args?`
: calls the specified XOTcl method without the need of using "`[list [self] methodName ...]`".

`::xotcl::alias class|obj methodName ?-objscope? ?-per-object? cmdName`
: can be used to register a predefined C-implemented Tcl command as method `methodName`. The option `-objscope` has the same meaning as for forwarder (instance variables of the calling object appear in the local scope of the Tcl command), `-per-object` has the same meaning as for the method `method` (when used on a class, the method is registered for the class object only, but not for the instances). This command can be used to bootstrap xotcl (when e.g. no methods are available).

::xotcl::configure filter ?on|off?
: allows to turn on or off filters globally for the current interpreter. By default, the filter state is turned off. This function returns the old filter state. This function is needed for the serializer that is intended to serialize the objects classes independent of filter settings.

`::xotcl::configure softrecreate ?on|off?`
: allows to control what should happen, when an object / a class is recreated. Per default it is set off, which means that the object/class is destroyed and all relations (e.g. subclass/superclass) to other objects/classes are destroyed as well. If `softrecreate` is set, the object is reseted, but not destroyed, the relations are kept. This is important, when e.g. reloading a file with class definitions (e.g. when used in

OpenACS with file watching and reloading). With `softrecreate` set, it is not necessary to recreate dependent subclasses etc.

Example: e.g. there is a class hierarchy A softrecreate set, a reload of B means first a destroy of B, leading to A softrecreate is set, the structure remains unchanged.

`::xotcl::finalize`

Delete all XOTcl objects and classes and free all associated memory.

This command has the only purpose to delete all objects and classes of an interpreter in a multi-threaded environment at a safe time.

Background: when XOTcl is used in a threaded environment such as for example in AOLserver, one has to take care that the deletion of objects and classes happens in a safe environment, where the XOTcl destructors (destroy methods) are still able to run. Without ::xotcl::finalize the deletion happens in Tcl_FinalizeThread(), after thread cleanup (where e.g. the thread local storage is freed). This can lead to memory leaks in AOLserver, which allocates e.g. some structures on demand, but since this happens after cleanup, it will leak. A simple ns_log in a destructor might lead to this problem. The solution is to call ::xotcl::finalize in the "delete trace" in AOLserver (as it happens in OpenACS).

Note, that ::xotcl::finalize is not intended for application programs.

Class: *::xotcl::Slot*

Class: Class
Heritage: Object

Description: A slot is a meta-object that manages property-changes of objects. A property is either an attribute or a role of an relation (e.g. in system slots). The predefined system slots are `class`, `superclass`, `mixin`, `instmixin`, `filter`, `instfilter`. These slots appear as methods of Object or Class.

The slots provide a common query and setting interface. Every multivalued slot provides e.g. a method `add` to add a value to the list of values, and a method `delete` which removes it. See for example the documentation of the slot mixin.

Parameters:

-name	Name of the slot to access from an object the slot
-domain	domain (object or class) of a slot on which it can be used
-multivalued	boolean value for specifying single or multiple values (lists)
-defaultmethods	list of two elements for specifying which methods are called per default, when no slot method is explicitly specified
-manager	the manager object of the slot (per default [self])
-per-object	

specify whether a slot should be used per class or per object; note that there is a restricted usage if applied per class, since defaults etc, work per initialization

For more details, consult the tutorial.

Class: *Attribute*

Class: Class
Heritage: ::xotcl::Slot

Description: Attribute slots are used to manage the setting and querying of instance variables. Parameters:

-default	specify a default value
-type	specify the type of a slot
-initcmd	specify a Tcl command to be executed when the value of the associated variable is read the first time; allows lazy initialization
-valuecmd	specify a Tcl command to be executed whenever the variable is read
-valuechangedcmd	specify a Tcl command to be executed whenever the variable is changed

Example of a class definition with three attribute slots:

```
Class Person -slots {
  Attribute name
  Attribute salary -default 0
  Attribute projects -default {} -multivalued true
}
Person p1 -name "John Doe"
```

The slot parameters `default`, `initcmd` and `valuecmd` have to be used mutually exclusively. For more details, consult the tutorial.

Class: *Class*

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

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 an uninitialized XOTcl object or class. Alloc is used by the method `create` to allocate the object. Note that `create` also calls as well `configure` and `init`

to initialized the object. Only in seldom cases the programmer may want to suppress the create mechanism and just allocate uninitiaized objects via alloc.

Return: new class name

- **allinstances**

  Description: Compute all immediate and indirect instances of a class

  Return: fully qualified list of instances

- **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. The method `create` is responsible for allocating and initializing objects. The method can be overloaded e.g. in a metaclass if other initialization behavior is wanted.

  The standard behavior of `create` is as follows:

  1. Call the method `alloc` to create an uninitialized object.
  2. Call the method `searchDefaults` to set default values for instance attributes-
  3. Call the method `configure` to configure the object with the values provided at object creation time. The method `configure` interprets the arguments with leading dashes as method calls.
  4. Call the method `init` to allow initialization by the class. The argument passed to init are the values from the passed argument list containing the arguments up to the first '-'.

  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). Finally the constructor `init` is called on the object with all arguments up to the first '-' arg.

  The `create` method is often 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 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 ?-closure? ?pattern?`: Returns a list of the instances of the class. If `-closure` is specified, the resultet contains as well the instances of subclasses. If `pattern` is specified and it contains wildcards, all matching instances are returned. If `pattern` does not contain wildcards, either the fully qualified name is returned, or empty, if no match exists.
- `ClassName info instargs method`: Returns the arguments of the specified instproc (instance method).
- `ClassName info instbody method`: Returns the body of the specified instproc (instance 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 instdefault method arg var`: Returns 1 if the argument `arg` of the instproc (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.
- `objName info instforward ?-definition name? ?pattern?`: Returns the list of instforwarders. One can call this method either without the optional arguments, or with the `pattern` or with `-definition name`. When the `pattern` is specified only the matching instforwarders are returned. When the `definition` option is used together with a name of a isntforwarder, the definition of the instforwarder with all flags is returned in a way that can be used e.g. for registering the instforwarder on another class.
- `ClassName info instinvar`: Returns class invariants.
- `ClassName info instmixin ?pattern?`: Returns the list of instmixins of this class. If `pattern` is specified and it contains wildcards, all matching mixin classes are returned. If `pattern` does not contain wildcards, either the fully qualified name is returned, or empty, if no match exists.
- `ClassName info instmixinof ?-closure? ?pattern?`: Returns the list of classes, into which this class was mixed in via instmixin. This is the inverse function of `ClassName info instmixin`. If `-closure` is specified, also the classes are returned, for which the class is indirectly mixed in via instmixin. If `pattern` is specified and it contains wildcards, all matching mixin classes are returned. If `pattern` does not contain wildcards, either the fully qualified name is returned, or empty, if no match exists.
- `ClassName info instnonposargs methodName`: returns list of non-positional args of methodName

- `objName info instparametercmd ?pattern?`: Returns a list of registered instparametercmds the class (or empty if there are none). If `pattern` is specified, only the matching instparametercmds are returned.
- `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 mixinof ?-closure? ?pattern?`: Returns the list of classes, into which this class was mixed in via per object mixin. This is the inverse function of `Object info mixin`. If `-closure` is specified, also the classes are returned, for which the class is indirectly mixed in as a per-object mixin. If `pattern` is specified and it contains wildcards, all matching mixin classes are returned. If `pattern` does not contain wildcards, either the fully qualified name is returned, or empty, if no match exists.
- `ClassName info parameter`: Returns parameter list.
- `ClassName info subclass ?-closure? ?pattern?`: Returns a list of all subclasses of the class. If `-closure` is specified, the result contains as well the subclasses of the subclasses. If `pattern` is specified and it contains wildcards, all matching subclasses are returned. If `pattern` does not contain wildcards, either the fully qualified name is returned, or empty, if no match exists.
- `ClassName info superclass ?-closure? ?superclassname?`: Returns a list of all super-classes of the class. If `-closure` is specified, the result contains as well the superclasses of the superclasses. If `pattern` is specified and it contains wildcards, all matching superclasses are returned. If `pattern` does not contain wildcards, either the fully qualified name is returned, or empty, if no match exists.

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 `assign`, `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** *methodName ?options? ?callee? ?args?*

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

  ?options?: -objscope, -methodprefix string, -default names, -earlybinding, -verbose

  ?callee?: named of the called command or object

  ?args?: arguments

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

  - %proc: substituted by name of the forwarder method
  - %self: substitute by name of the object
  - %1: substitute by first argument of the invocation
  - {%@POS value}: substitute the specified value in the argument list on position POS, where POS can be a positive or negative integer or `end`. Positive integers specify the position from the begin of the list, while negative integer specify the position from the end.
  - {%argclindex LIST}: take the *n*th argument of the specified list as substitution value, where *n* is the number of arguments from 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 (only in connection with %1)
  - `-earlybinding`: look up the function pointer of the called Tcl command at definition time of the forwarder instead of invocation time. This option should only be used for calling C-implemented Tcl commands, no procs etc.);
  - `-verbose`

  : print the substituted command to stderr before executing

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 `assign`, `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?": optional 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 by two additional arguments. Therefore, to specify only post-assertions an empty pre-assertion list must be given. All assertions are a list of ordinary Tcl conditions.
  When instproc is called with an empty argument list and an empty body, the specified instproc is deleted.

  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: Methods called upon recreation of an object. Recreate is called, when an object/class is created, but a same-named object/class exists already. "recreate" is not called, when an object is trying to be recreated as a class or vice versa. In these cases, recreating is realized via destroy+create. The Methods "recreate" 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`. The method returns the current value of `superclass`, when it is called without arguments.

Return: if `classList` is not specified return superclass(es), otherwise empty

- **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

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.

  This method is called on mixin/instmixin definition calls, istype, ismixin, class, superclass and parameterclass

  Return: empty string

Class: *Object*

Class: Class

Procs/Instprocs: abstract, append, array, autoname, check, class, cleanup, configure, contains, 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, subst, trace, unset, uplevel, upvar, volatile, 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 methodName arglist*

  Arguments: **methtype**: instproc or proc

  methodName: 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?

Description: Changes the class of an object dynamically to `newClass`. The method returns the current value of `class`, when it is called without arguments.

Return: if `newClass` is not specified return class, otherwise empty

- **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 '-' (dash) 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 `init` during initialization and recreation. In the following example, the variable set is called via configure before `init`:

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

The method `configure` can be called with the dash-notation at arbitrary times:

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

Note, that if '-' is followed by a numerical, the arument is interpreted as a negative number (and not as a method). If a value of a method called this way starts with a "-", the call can be placed safely into a list (e.g. "Class c [list -strangearg -a-] -simplearg 2").

See also create.

Return: number of the skipped first arguments

- **contains** *?-withnew? ?-object? ?-class? cmd*

Arguments: **?-withnew?**: Option to overload new to create new objects within the specified object. Per default, this option is turned on.

?-object?: object, in which the new objects should be created. The default is the object, for which `contains>/tt> was called.`

?-class?: In combination with option `-object`: If the specified object does not exist, create it from the specified class. The default is `::xotcl::Object`

cmd: Tcl command to create multiple objects

Description: This method can be used to create nested object structures with little syntactic overhead. The method changes the namespace to the specified object and creates objects there. Optionally, a different object scope can be specified and creating new objects in the specified scope can be turned off. The following command creates a three rectangles, containing some points.

```
Class Point -parameter {{x 100} {y 300}}
Class Rectangle -parameter {color}

Rectangle r0 -color pink -contains {
  Rectangle r1 -color red -contains {
```

```
    Point x1 -x 1 -y 2
    Point x2 -x 1 -y 2
  }
  Rectangle r2 -color green -contains {
    Point x1
    Point x2
  }
}
```

The resulting object structure looks like in the folloing example (simplified).

```
::r0
::r0::r1
::r0::r1::x1
::r0::r1::x2
::r0::r2
::r0::r2::x1
::r0::r2::x2
```

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 `assign`, `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** *methodName ?options? ?callee? ?args?*

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

?options?: -objscope, -methodprefix string, -default names, -earlybinding, -verbose

?callee?: named of the called command or object

?args?: arguments

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

- %proc: substituted by name of the forwarder method
- %self: substitute by name of the object
- %1: substitute by first argument of the invocation
- {%@POS value}: substitute the specified value in the argument list on position POS, where POS can be a positive or negative integer or `end`. Positive integers specify the position from the begin of the list, while negative integer specify the position from the end.
- {%argclindex LIST}: take the *n*th argument of the specified list as substitution value, where *n* is the number of arguments from 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 (only in connection with %1)
- `-earlybinding`: look up the function pointer of the called Tcl command at

definition time of the forwarder instead of invocation time. This option should only be used for calling C-implemented Tcl commands, no procs etc.);

♦ `-verbose`

: print the substituted command to stderr before executing

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 proc (object specific method).
  - `objName info body method`: Returns the body of the specified proc (object specific method).
  - `objName info class`: Returns the name of the class of the current object.
  - `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 proc (object specific 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 forward ?-definition name? ?pattern?`: Returns the list of forwarders. One can call this method either without the optional arguments, or with the `pattern` or with `-definition name`. When the `pattern` is specified only the matching forwarders are returned. When the `definition` option is used together with a name of a forwarder, the definition of the forwarder with all flags is returned in a way that can be used e.g. for registering the forwarder on another object.
  - `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 methods`: Returns the list of all methods 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 ?-order? ?-guard? ?pattern?`: Returns the list of mixins of the object. With `-order` modifier the order of mixins (whole hierarchy) is printed. If `-guard` is specified, the mixin guards are returned. If `pattern` is specified and it contains wildcards, all matching mixins are returned. If `pattern` does not contain wildcards, either the fully qualified name is returned, or empty, if no match exists.
- `objName info nonposargs methodName`: Returns non-positional arg list of methodName
- `objName info parametercmd ?pattern?`: Returns a list of registered parametercmds the object (or empty if there are none). If `pattern` is specified, only the matching parametercmds are returned.
- `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 precedence ?-intrinsic? ?pattern?`: Returns all classes in the precedence order from which the specified object inherits methods. If the flag `-intrinsic` is specified only the intrinsic classes (from the class hierarchy) are specified. If the flag is not specified, the returned list of classes contains the mixin and instmixin classes as well as the classes of the superclass chain in linearized order (i.e., duplicate classes are removed). If the pattern is specified, only matching classes are returned.
- `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**: name of instance variable

?v2...vn?: optional other names for instance 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 `assign`, `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". Note that move is currently implemented as a copy plus subsequent destroy operation.

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?: optional 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 specified by two additional arguments. Therefore, to specify only post-assertions an empty pre-assertion list must be given. All assertions are a list of ordinary Tcl conditions.
When instproc is called with an empty argument list and an empty body, the specified instproc is deleted.

Return: empty string

- **procsearch** *procName*

Arguments: **procName**: simple proc name

Description: Search which method should be invoked for an object and return the fully qualified name of the method as a list in proc qualifier format: 'objName|className proc|instproc|forward|instforward|parametercmd|instparametercmd|cmd|instcmd methodName'. The proc qualifier format reports the command used to create the method. The only exception is instcmd and cmd, which refer to commands implemented in C. E.g.,

```
o procsearch set
```

returns

```
::xotcl::Object instcmd 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

- **subst** *options string*

  Arguments: **options**: ?-nobackslashes? ?-nocommands? ?-novariables?

  string: string to be substituted

  Description: Perform backslash, command, and variable substitutions in the scope of the given object (see documentation of Tcl command with the same name for details).

  Return: substituted string

- **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** *?-nocomplain? v1 ?v2...vn?*

  Arguments: **?-nocomplain?**: possible error messages are suppressed

  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

  ```
  uplevel [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 variable 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

- **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

- **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.
