The TCL Architecture of Objects (TAO)

Developer's Manual

Written by: Sean Woods

Table of Contents

1 Porting code from [Incr Tcl]

Two special keywords **public** and **private** are present to facilitate the porting of [Incr Tcl] code. Calls to “public method” are redirected to they keyword “method”. “public variable” becomes a symbol, and likewise with private. Note that TAO does not honor data or method hiding.

All methods are run at the scope of the final object in TAO, no matter what class they are defined in. Users of [Incr Tcl] have to take into account when they call a method defined in a parent, that parent method can only see what was defined up to the point the parent class was created. Under TAO, you can reference non-existant methods and variables in a base object, knowing that they will be defined later on. This keeps you from having to make an IPC call from an object to itself.

Example 1 – TAO methods see all and *Example 2 – [Incr Tcl] has to use a workaround to call methods defined later* demonstrate the difference between TAO’s scope and [Incr Tcl]’s scope.

```
tao::class ::foo {
   method whip it {
      return [crack $it]
   }
}

tao::class ::bar {
   inherit ::foo
   method crack it { … }
   method go it {
       return [whip $it]
  }
}

::bar none
none go nuts
```

Example 1 – TAO methods see all

```
itcl::class ::foo {
   public method whip it {
      return [crack $it]
   }
   public method whip_fix it {
      return [$this crack $it]
   }
}

itcl::class ::bar {
   inherit ::foo
   public method crack it { … }
   public method go it {
       return [whip $it]
  }
}

::bar none
none go nuts
```

DIE DIE DIE: unknown command crack

Example 2 – [Incr Tcl] has to use a workaround to call methods defined later

TAO constructors do not take arguments. See section 2.1 for more details about the TAO constructor.

2 Methods and Procedures

2.1 constructor

Syntax: **constructor** *body*

constructor *prebody body*

Example:

```
::tao::class foo {
    constructor {
       set foo bar
       if [cget table table] {
           error “No value for table given in object definition.”
       }
       build_internal_state
    }
}
```

Description: Declare a constructor.

Constructor declares a script that is run during the creation process of an object. Constructors are treated in the same manner as a mesh. If two bodies are given, the first body is inserted before the main bodies of the ancestors run. The second body is meshed as normal.

Note that constructors in TAO do not take arguments. All new TAO objects take a TK style key/value list for initial values. It is the constructor’s job to detect missing data and throw an error.

2.2 destructor

Syntax: **destructor** *body*

Example:

```
::tao::class foo {
   destructor {
     give_away_toys
   }
}
```

Description: Declare a destructor.

Destructor specifies a script that is run during the destruction sequence of an object. Internally destructors are treated like a mesh.

2.3 consecution

Syntax: **consecution** *procname arglist body*

consecution *procname arglist prebody body*

Example: See: ***Error! Reference source not found.***: ***Error! Reference source not found.***

Description: Declare a consecution type of method.

2.4 mesh

Syntax: **mesh** *procname arglist body*

mesh *procname arglist prebody body*

Example: See: ***Error! Reference source not found.***: ***Error! Reference source not found.***

Description: Declare a mesh type of method.

2.5 method

Syntax: **method** procname arglist body

Example:

```
::tao::class ::foo {
    method froody {a b} {
       return [expr $a + $b]
   }
}
```

Description: Declare a method, along with argument list and body.

2.6 proc

Syntax: **proc** procname arglist body

Example:

```
::tao::class ::foo {
   proc froody {a b} {
       return [expr $a + $b]
   }
}
```

Description: Declare a procedure, along with argument list and body. Procedures differ from methods in that they do not get the state of the object.

3 Variable Declaration

3.1 hash

Syntax: **hash** *variablename ?defaultvalue?*

Example:
```
::tao::class ::foo {
    hash fooray {0 zero 1 one 2 two}

    method dump {} {
       return [array get fooray]
    }
}
```

Description: Declares a non-volatile array for use within methods.

Hash works the same way as symbol, only it ensures that the various handlers in the system treat the symbol as an array, not a variable.

3.2 symbol

Syntax: **symbol** *variablename ?defaultvalue?*

Example:
```
::tao::class ::foo {
     symbol foo bar
     method dump {} {
          return $bar
     }
}
```

Description: Declares a non-volatile variable for use within methods.

Symbol is the equivalent to a “public variable” in [Incr Tcl]. For backwards compatibility with [Incr Tcl] calls to “public variable” are mapped to symbol.

3.3 static

Syntax: **static** *variablename ?defaultvalue?*

Example:
```
::tao::class ::foo {
    static foo bar
    method dump {
         return $bar
    }
    method setfoo {value} {
        cset foo $value
    }
    method nosetfoo {value} {
         set foo $value
    }
}

# Example
::foo object
object dump
> bar
object nosetfoo another_value
object dump
> bar
object setfoo another_value
object dump
> another_value
```

Description: Static is designed to allow developers to build in constants within an object that are immune to change from within the methods of the object.

that are immune to change from within the methods of the object.

3.4 initvalues

Syntax: **initvalues** *keyvaluelist*

Example:

```
::tao::class ::foo {
    initvalues {foo a bar b baz c}
}
::foo object
object cget bar
> b
```

Description: Initvalues allows the developer to load a pile of constants into an object in one shot. Values are input as list of key/value pairs. If the key has been declared as a symbol earlier in the class' definition this value will replace the declared initial value.

Unless the variables have been declared as symbols or statics they can only be accessed with the object's cget or cset functions.

4 Standard Methods

All standard methods are defined in the class ::tao::root. ::tao::root is inherited by all classes under TAO whether it is explicitly stated or not.

4.1 chain

Syntax: **Chain**

Example: Chain

Description: Call the immediate ancestor to this class' method definition. Use of this keyword is only intended as a baby step for porting from [Incr Tcl]. Use of this keyword in new code is discouraged.

4.2 cget

Syntax: **cget** *fieldname varname*

cget *fieldname*

Example:

```
cget foo
> 10

…

if [cget foo bar] {
   # handle a non-configured value
   set bar 10
    cset bar $bar
}
return $bar
```

Description: Retrieve variable data from an object. Any leading dashes (-) are stripped from fieldname to be compatible with Tk style arguments.

If *varname* is not given the system simply returns the value (if any) for the field specified.

If *varname* is given, the system will feed the value (if any) for fieldname into the variable given. The call will return false if a value was already preset in the object, or true if it was not.

4.3 configure

Syntax: **configure** *?-argname ?value? ?-argname value? …*

configure *-argname*

configure

Example: ::object configure –foo bar

Description: An [Incr Tcl]/Tk compatible means to pass and retrieve data from an object

4.4 Container

Syntax: **Container** *handle*

Example: Container wiki

Description: Return and object reference for the container object specified in *handle*

4.5 cset

Syntax: **cset** *key value ?key value?…*

Example: cset foo bar

Description: Inputs data into the internal state of the object as given by the key/value pairs. Note: this is the one way to alter the value of a static variable.

4.6 dlearn

Syntax: **Dlearn** *object*

Example: ::foo dlearn ::baz

Description: Duplicate all variable data from one object to another. Certain key variables like globalName, containerObj, and nodeId are excluded.

4.7 Handle

Syntax: **Handle**

Example: ::foo Handle

Description: Return the tao handle for an object reference.

4.8 isa

Syntax: **isa** *classname*

Example: ::foo isa ::node

Description: Returns true if an object inherits properties from *classname*. False otherwise

4.9 morph

Syntax: **morph** *classname*

Example: ::foo morph ::node::other

Description: Change an object from one class to another. Note that the constructor for the new class is not run.

5 Node/Container Methods

Containers and node objects represent a two special family of objects in TAO. Containers act as a gateway with which to access node objects. The most straightforward example would by the relationship between an object that represents an SQL table, and the objects that represent records within that SQL table.

Containers and Nodes are two special form of object. Essentially Containers spawn Nodes and the two maintain a relationship throughout the lifecycle of the node object.

5.1 /container

Syntax: **/container**

Example: [::tao::/node red-2] /container

Description: Returns an object reference to a node's container. This is designed as a reply for outside calls. A static variable *containerObj* stores the same value for use within methods.

5.2 /node

Syntax: **/node** *nodeid*

Example: [::tao::Container wiki] /node 5

Description: Returns an object reference to a record node.

5.3 buildInternalState

Syntax: *N/a*

Example:

```
::tao::class ::foo {
    inherit ::tao::container
    static walk
    constructor {
         if [cget walk style] {
             error “I have no walk”
         }
     }
     method walk {} {
           return “Like a(n) $walk”
     }
}
::tao::class ::bar {
   inherit ::foo
   mesh buildInternalState {} {
         cset walk Egyption
   }
}
::foo a
> ERROR: I have no walk
::foo a –walk Assyrian
a walk
> Walk like a(n) Assyrian
::bar b
b walk
> Walk like a(n) Egyptian
```

Description: Nodes and container classes have a special method that is called during the baseclass constructor called buildInternalState. This method should never take an argument, and always mesh with its parent class.

buildInternalState is called as the first command in the constructor, so it is a helpful tool for objects to auto-detect settings, sanity check parameters, etc.

5.4 defaultNodeClass, defaultNewClass, defaultBaseClass

Syntax: *N/a*

Example:

```
::tao::class ::foo {
    inherit ::tao::container
    static walk

    proc defaultNodeClass {} {
        return open
    }
    proc defaultBaseClass {} {
         return ::widget
    }
}
::tao::class ::widget {
   inherit ::tao::node
   method state {} {
       return n/a
   }
   method open {} {
       morph open
   }
   method close {} {
       morph closed
   }
}
::tao::class ::widget::open {
   inherit ::widget
   method state {} {
       return open
   }
   method open {} {
       error “I’m already open”
   }
}
::tao::class ::widget::closed {
   inherit ::widget
   method state {} {
      return closed
   }
   method close {} {
      error “I’m already closed”
   }
}

::foo cObj
set aObj [cObj spawn_node A]
$aObj state
> open
$aObj close
$aObj state
> closed
$aObj close
> ERROR: I’m already closed
$aObj open
$aObj state
> open
```

Description: TAO allows containers to control what kinds of nodes they generate by overriding any one of three classes. Each call takes no arguments, and returns a string.

defaultBaseClass can provide a base class that other class references can be built around

defaultNodeClass is what specific class should be used if RecordClass returns nothing.

defaultNewClass is called when the container knows it is creating a new object. This is particularly helpful for database objects where records pass through various stages from creation to completion.

5.5 nodeAlias

Syntax: *N/a*

Example:

```
::tao::class ::foo {
    inherit ::tao::container

    symbol objectList

    method nodeAlias nodeid {
        switch $nodeid {
            first { return [lindex [lsort $objectList 0]] }
            last  { return [lindex [lsort $objectList end]] }
        }
        return $nodeid
    }
}
```

Description: nodeAlias maps an arbitrary string to an internal record number. This is helpful for UI systems where the interface may call for a record by something other than it’s ID string.

5.6 recordClass

Syntax: *N/a*

Example:

```
::tao::class ::store {
    inherit ::tao::container

    static table
    static primary_key
    static classfield

    method defaultBaseClass {} {
       return ::pet
    }

    method recordClass nodeid {
        set node [nodeAlias $nodeid]
        set class [sqlSelect “$classfield from $table where
${primary_key}=$node”]
        return $class
    }
}
::tao::class ::pet {
    inherit ::tao::node ::animal
}
foreach {alias baseanimal} {
  kitty cat
  hammy hampster
  puppy dog
  doggie dog
  fishy fish
  goldy fish
} {
  ::tao::class ::pet::${alias} [list inherit ::pet [::tao::class join ::animal
$baseanimal] ; static whatami $alias]
}
```

Description: recordClass allows a container to intelligently map a record id to a proper class. It takes one argument, the node ID of the object to be created. It should return a string, or an empty list for no answer.

5.7 Train

Syntax: *N/a*

Example:

```
::tao::class ::foo {
    inherit ::tao::container
    mesh Train object {
        $object cset allyourbase “Are now belong to us”
    }
}
```

Description: Train is a standard method in an object that conditions newly spawned objects. Normally nodes dlearn from their containers what they need to know about the environment, but there may be cases where you need the container to impart some special knowledge that is not covered by one of the other mechanisms.

Train is automatically called immediately after the object’s constructor resolves.