

Protobj

Prototype-Delegation Object Model in Scheme

Version 0.2
2005-06-19

Neil W. Van Dyke
neil@neilvandyke.org

<http://www.neilvandyke.org/protobj/>

Copyright © 2005 Neil W. Van Dyke. This program is Free Software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version. This program is distributed in the hope that it will be useful, but without any warranty; without even the implied warranty of merchantability or fitness for a particular purpose. See <http://www.gnu.org/copyleft/lesser.html> for details. For other license options and consulting, contact the author.

1 Introduction

Protobj is a Scheme library that implements a simple prototype-delegation object model, somewhat similar to that of **Self**, and also related to those of **SLIB object** and **OScheme**. Protobj was written mainly as a **syntax-rules** learning exercise, but also because people ask about prototype object models for Scheme from time to time. Like most object systems, Protobj should be regarded as an amusement. The Protobj library defines both a verbose set of procedures, and terse special syntax.

Protobj is based on objects with named slots that can contain arbitrary values. Object have immediate slots, and single parent objects from which additional slots are inherited. When setting in a child object a slot inherited from the parent, a new immediate slot is created in the child so that the parent is unaffected and the slot is no longer inherited.

Methods are simply closures stored in slots. When a method is applied, the first term of the closure is the receiver object. Unlike Self, getting getting the contents of the slot is distinguished from invoking a method contained in the slot. This distinction was made due to the way first-class closures are often used in Scheme.

An object is cloned by invoking the `clone` method. The default root object's `clone` method creates a new child object without any immediate slots, rather than copying any slots. This behavior can be overridden to always copy certain slots, to copy immediate slots, or to copy all inherited slots. An overriding `clone` method can be implemented to apply its parent's `clone` method to itself and then set certain slots in the new child appropriately.

Protobj requires R5RS, SRFI-9, SRFI-23, and SRFI-39.

2 Tour

The following is a quick tour of Protobj using the terse special syntax.

1. Bind **a** to the new object that is created by cloning the default root object (% is special syntax for invoking the `clone` method):

```
(define a (%))
```
2. Verify that **a** is an object and that **a**'s parent is the default root object:

```
(object? a) ⇒ #t  
(eq? (^ a) (current-root-object)) ⇒ #t
```
3. Add to **a** a slot named **x** with value 1:

```
(! a x 1)
```
4. Get **a**'s slot **x**'s value:

```
(? a x) ⇒ 1
```
5. Bind **b** to a clone of **a**:

```
(define b (% a))
```
6. Get **b**'s slot **x**'s value, which is inherited from **a**:

```
(? b x) ⇒ 1
```
7. Set **a**'s slot **x**'s value to 42, and observe that **b** inherits the new value:

```
(! a x 42)  
(? a x) ⇒ 42  
(? b x) ⇒ 42
```
8. Set **b**'s slot **x**'s value to 69, and observe that **a** retains its own **x** value although **b**'s value has been changed:

```
(! b x 69)  
(? a x) ⇒ 42  
(? b x) ⇒ 69
```
9. Add to **a** an `xplus` slot containing a closure that implements a method of the object:

```
(! a xplus (lambda ($ n) (+ (? $ x) n)))
```
10. Apply the method to the **a** and **b** objects (**b** inherits any new slots added to **a**):

```
(@ a xplus 7) ⇒ 49  
(@ b xplus 7) ⇒ 76
```
11. Observe the shorthand syntax for applying methods to an object multiple times, with the syntax having the value of the lastmost application:

```
(@ a (xplus 1000) (xplus 7)) ⇒ 49
```
12. Bind to **c** an object that clones **a** and adds slot **y** with value 101:

```
(define c (% a (y 101)))
```
13. Get the values of both the **x** and **y** slots of **c**:

```
(? c x y) ⇒ 42 101
```
14. Finally, bind **d** to a clone of **a** that overrides **a**'s **x** slot:

```
(define d (% a (x 1) (y 2) (z 3)))  
(? d x y z) ⇒ 1 2 3
```

3 Basic Interface

The basic interface of Protobj is a set of procedures.

`object? x` [Procedure]
Predicate for whether or not `x` is a Protobj object.

`object-parent obj` [Procedure]
Yields the parent object of object `obj`.

`object-set! obj slot-symbol val` [Procedure]
Sets the slot identified by symbol `slot-symbol` in object `obj` to value `val`.

`object-get obj slot-symbol` [Procedure]
Yields the value of slot named by symbol `slot-symbol` in object `obj` (immediate or inherited). If no slot of that name exists, an error is signaled.

`object-get obj slot-symbol noslot-thunk` [Procedure]
Yields the value of slot named by symbol `slot-symbol` in object `obj` (immediate or inherited), if any such slot exists. If no slot of that name exists, then yields the value of applying closure `noslot-thunk`.

`object-apply obj slot-symbol { arg }*` [Procedure]
Applies the method (closure) in the slot named by `slot-symbol` of object `obj`. The first term of the method is `obj`, and one or more `arg` are the remaining terms. If no such slot exists, an error is signaled.

`object-apply/noslot-thunk obj noslot-thunk slot-symbol { arg }*` [Procedure]
Like `object-apply`, except that, if the slot does not exist, instead of signalling an error, the value is the result of applying `noslot-thunk`.

`object-raw-clone/no-slots-copy obj` [Procedure]

`object-raw-clone/copy-immed-slots obj` [Procedure]

`object-raw-clone/copy-all-slots obj` [Procedure]

These procedures implement different ways of cloning an object, and are generally bound as `clone` methods in root objects. `/no-slots-copy` does not copy any slots, `/copy-immed-slots` copies immediate slots, and `/copy-all-slots` copies all slots including inherited ones.

`current-root-object` [Parameter]
Parameter for the default root object. The initial value is a root object that has `object-raw-clone/no-slots-copy` in its `clone` slot.

4 Terse Syntax

Since Protobj's raison d'être was to play with syntax, here it is. Note that slot names are never quoted.

`^ obj` [Syntax]
Parent of *obj*.

`! obj slot val` [Syntax]

`! obj (slot val) ...` [Syntax]
Sets object *obj*'s slot *slot*'s value to *val*. In the second form of this syntax, multiple slots of *obj* may be set at once, and are set in the order given.

`? obj { slot }+` [Syntax]
Yields the values of the given *slots* of *obj*. If more than one *slot* is given, a multiple-value return is used.

`@ obj slot { arg }*` [Syntax]

`@ obj { (slot { arg }*) }+` [Syntax]
Applies *obj*'s *slot* method, with *obj* as the first term and *args* as the remaining terms. In the second form of this syntax, multiple methods may be applied, and the value is the value of the last method application.

`% [obj { (slot val) }*]` [Syntax]
Clones object *obj*, binding any given *slots* to respective given *vals*.

5 Tests

The Protobj test suite can be enabled by editing the source code file and loading [Testeez](#).

History

Version 0.2 — 2005-06-19

Fixed bug in `%protobj:apply*` (thanks to Benedikt Rosenau for reporting).
Changed `$` syntax to `?`, so that `$` could be used for “self” in methods. Documentation changes.

Version 0.1 — 2005-01-05

Initial release.