# Recipes for PDSC #5: More Elaborate Structures

*by Tom Christiansen*
**< tchrist@perl.com >**

**release 0.2 (tested)**
**Monday, 2 October 1995**

---

## Declaration of MORE ELABORATE RECORDS:

Here's a sample showing how to create and use a record whose fields are of many different sorts:

```
$rec = {
    STRING  => $string,
    LIST    => [ @old_values ],
    LOOKUP  => { %some_table },
    FUNC    => \&some_function,
    FANON   => sub { $_[0] ** $_[1] },
    FH      => \*STDOUT,
};

print $rec->{STRING};

print $rec->{LIST}[0];
$last = pop @ { $rec->{LIST} };

print $rec->{LOOKUP}{"key"};
($first_k, $first_v) = each %{ $rec->{LOOKUP} };

$answer = &{ $rec->{FUNC} }($arg);
$answer = &{ $rec->{FANON} }($arg1, $arg2);

# careful of extra block braces on fh ref
print { $rec->{FH} } "a string\n";

use FileHandle;
$rec->{FH}->autoflush(1);
$rec->{FH}->print(" a string\n");
```

## Declaration of a HASH OF COMPLEX RECORDS:

```
%TV = (
   "flintstones" => {
       series   => "flintstones",
       nights   => [ qw(monday thursday friday) ];
       members  => [
           { name => "fred",    role => "lead", age  => 36, },
```

```
            { name => "wilma",   role => "wife", age  => 31, },
            { name => "pebbles", role => "kid", age  =>  4, },
        ],
    },

    "jetsons"     => {
        series   => "jetsons",
        nights   => [ qw(wednesday saturday) ];
        members  => [
            { name => "george",  role => "lead", age  => 41, },
            { name => "jane",    role => "wife", age  => 39, },
            { name => "elroy",   role => "kid",  age  =>  9, },
        ],
     },

    "simpsons"    => {
        series   => "simpsons",
        nights   => [ qw(monday) ];
        members  => [
            { name => "homer", role => "lead", age  => 34, },
            { name => "marge", role => "wife", age => 37, },
            { name => "bart",  role => "kid",  age  =>  11, },
        ],
     },
  );
```

# Generation of a HASH OF COMPLEX RECORDS:

```
# reading from file
# this is most easily done by having the file itself be
# in the raw data format as shown above.  perl is happy
# to parse complex datastructures if declared as data, so
# sometimes it's easiest to do that

# here's a piece by piece build up
$rec = {};
$rec->{series} = "flintstones";
$rec->{nights} = [ find_days() ];

@members = ();
# assume this file in field=value syntax
while () {
    %fields = split /[\s=]+/;
    push @members, { %fields };
}
$rec->{members} = [ @members ];

# now remember the whole thing
$TV{ $rec->{series} } = $rec;

###########################################################
# now, you might want to make interesting extra fields that
# include pointers back into the same data structure so if
# change one piece, it changes everywhere, like for examples
# if you wanted a {kids} field that was an array reference
# to a list of the kids' records without having duplicate
# records and thus update problems.
###########################################################
```

```
foreach $family (keys %TV) {
    $rec = $TV{$family}; # temp pointer
    @kids = ();
    for $person ( @{$rec->{members}} ) {
        if ($person->{role} =~ /kid|son|daughter/) {
            push @kids, $person;
        }
    }
    # REMEMBER: $rec and $TV{$family} point to same data!!
    $rec->{kids} = [ @kids ];
}

# you copied the list, but the list itself contains pointers
# to uncopied objects. this means that if you make bart get
# older via

$TV{simpsons}{kids}[0]{age}++;

# then this would also change in
print $TV{simpsons}{members}[2]{age};

# because $TV{simpsons}{kids}[0] and $TV{simpsons}{members}[2]
# both point to the same underlying anonymous hash table

# print the whole thing
foreach $family ( keys %TV ) {
    print "the $family";
    print " is on during @{ $TV{$family}{nights} }\n";
    print "its members are:\n";
    for $who ( @{ $TV{$family}{members} } ) {
        print " $who->{name} ($who->{role}), age $who->{age}\n";
    }
    print "it turns out that $TV{$family}{'lead'} has ";
    print scalar ( @{ $TV{$family}{kids} } ), " kids named ";
    print join (", ", map { $_->{name} } @{ $TV{$family}{kids} } );
    print "\n";
}
```