



## Programming Parrot: Parrot Magic Cookies

Jonathan Scott Duff  
duff@pobox.com

### A quick review

As Alberto Manuel Simões showed in “Programming Parrot” (TPR 2.3), Parrot is a register-based virtual machine with four register types: Integer, String, Number, PMC. There are thirty-two registers for each type (numbered 0 through 31), which I reference by a capital letter signifying the register type followed by the register number. For example, `s15` is String register number 15. A parrot program is lines of text where each line is one opcode with its arguments.

By using a slightly higher-level syntax called Parrot Intermediate Representation (PIR), I can obtain an arbitrary number of each register type by prefixing the register with a `$`. For example, the virtual register `$I51` is perfectly valid register even though parrot doesn't have fifty-one integer registers. PIR also provides for a more natural syntax for opcodes. Rather than saying `set I1, 0` to assign zero to the I1 register, I say `I1 = 0`. PIR also provides syntax for easily creating named variables and constants, subroutines, passing parameters to subroutines, accessing parameters by name, and so on.

### Now, on to business ...

Integers, strings, and arbitrary floating point numbers are common data types in most programming languages, but what's a “Parrot Magic Cookie” (PMC)? PMCs let Parrot handle more complicated

*To run the example code in this article, you'll need to get a copy of Parrot and build it for your system. For information on obtaining Parrot, see <http://www.parrotcode.org/>. Instructions for compiling Parrot are available in the Parrot distribution itself. All code examples in this article were tested with Parrot 0.4.5.*

structures and behaviors, hence the magic. Anything that I can't express using just integers, floating point numbers, or strings I can express with a PMC, such as arrays, hashes, and objects.

Parrot comes with many types of PMC that encapsulate common, useful behavior. This program that lists all of the PMC types built into Parrot:

```
.sub _ :main
  $I0 = 1          # first PMC
  loop:
    $I1 = valid_type $I0
    unless $I1 goto end_loop
    $S0 = typeof $I0 # PMC type name
    print $I0        # PMC number
    print "\t"
    print $S0        # PMC type name
    print "\n"
    inc $I0
    goto loop
  end_loop:
.end
```

In this program, I use the `typeof` opcode to determine the type of each PMC. Given an integer argument, `typeof` returns the name of the PMC type corresponding to that integer as a string. Most of the PMC type names give clues as to how I use them. Here's a table that gives a short description of several interesting and useful PMC types:

#### Perl one-liner

See what Perl sees by using the `B::Deparse` module. It parses your code, then unparses it to give you back the program Perl thinks it's doing:

```
$ perl -MO=Deparse program.pl
```