

Parrot Status Report

by Jonathan Scott Duff
duff@pobox.com



Parrot has been in development since late 2001. What do we have to show for the past 6 years of effort? What's in Parrot's future? When will it be complete? This article will hopefully give you some idea as to the answers or at least point you in the general direction of an answer.

■ What is Parrot? -----

For those who don't already know, Parrot is a virtual machine designed for running dynamic languages such as Perl, Python, Ruby, *etc.* It was initially conceived to be the underlying virtual machine for Perl 6 but its mission also includes other dynamic languages since the problems faced by those languages are very similar. Parrot is implemented in C, but currently to build Parrot you also need to have Perl installed.

■ Where is Parrot today? -----

As of this writing, Parrot has specifications and implementations for bytecode, namespaces, lexical variables, objects, events, exceptions, and basic IO. There are just-in-time compilers for x86, amd64, ia64, ppc, mips, hppa and arm processors. Parrot has been made to run on these platforms: Linux, Solaris, Mac OS X, FreeBSD, Windows, and cygwin. There are many languages that have a Parrot implementation. Parrot has even been embedded in a certain popular web server.

■ Low level bits: What makes Parrot tick -----

The following are available in Parrot to make various software items such as language implementations and execution environments. They may not all be completely implemented, but generally enough has been implemented to be useful. Indeed, several of the languages that target Parrot do use these features:

Garbage collection

Parrot utilizes a generational mark and sweep garbage collector to ensure timely destruction of objects while also minimizing the impact on performance.

Continuations

Parrot uses something called Continuation Passing Style for calling subroutines and other flow control items. Essentially a continuation is a snapshot of the execution environment that can be resumed at any time. Thus, at the end of a subroutine, a "return" is really just an execution of the continuation that was taken at the start of the subroutine with any values that the subroutine may want to return as arguments to the continuation.

See <http://www.intertwingly.net/blog/2005/04/13/Continuations-for-Curmudgeons> for more info on continuations and continuation passing style.

Parrot hides the explicit nature of continuations behind syntax for executing and returning from subroutines, but there are opcodes to take and call continuations when I need them.

Parrot Magic Cookies (PMC)

These are fundamental units of stuff in Parrot that aren't numbers or strings. PMCs are like objects in that they implement certain behavior and that behavior can be inherited by other PMCs. Language implementors can use PMCs to describe some detail of their language that Parrot may not natively provide.

Namespaces

Namespaces have been fully specified and implemented. These can be used to partition code or data into useful compartments. For instance, each language that targets Parrot has its own namespace so that multiple languages can interoperate in the same Parrot execution. Namespaces are hierarchical in nature so that languages that support their own idea of namespaces can also make direct use of Parrot's namespace mechanism.

Objects

Many dynamic programming languages are object-oriented in nature or have object-oriented features. Parrot provides an object model and primitive operations that allow these languages to implement the semantics appropriate for any object system. There is support for classes, attributes on those classes, methods on those classes, inheritance (single and multiple), multi-method dispatch, delegation, class composition (roles), object introspection, *etc.*

Events

In order to be compatible with event-driven programming, Parrot provides a standard interface for events that works in conjunction with a scheduler to fire actions in response to those events.

Exceptions

When something unusual happens in the course of execution, what is Parrot to do? One of the things it can do is throw an exception. There are facilities for throwing and catching exceptions, adding and removing exception handlers, querying whether or not an exception has been handled or not, and terminating execution as appropriate.