Home

Home page
Getting started
Facts Table
About Falcon
People at work
Licensing

Downloads

Documents

Extensions

Contacts

Site source

Donate

About the Falcon Programming Language

People ask me: "With so many languages around, why start to write another one?"

The answer is stunningly simple: there wasn't any language that completely satisfied the needs I had -- needs that are basic to a professional IT consultant writing performance-critical and heavily multithreaded applications.

Past experiences

I think the reason for this is that mainstream programming languages were born for very specific tasks, or simply for fun, or for academic research. They rarely were born to do the thing they were meant to: help IT experts to walk their life.

Two major exceptions had remarkable impact. The Dbase/clipper/XBase languages, born to write business oriented applications, made the IT scene for a long while until were killed by suicidal commercial policies. PERL was created precisely to parse files efficiently and effectively (it's acronym is Practical Extraction and Report Language), and also to help integrate other UNIX (text) processors. And since those were quite generic tasks and were also quite common needs for anyone works with IT, these languages had an excellent and long lasting success.

No other way

But long story short: there was simply no viable alternative. Languages born as stand-alone solutions, and then made embeddable, or simply seen mainly as stand-alone systems, were simply too closed to integrate well as drop-ins with existing huge applications. Worst of all, their requirements under multithreading environments were simply unreasonable. Also, I was concerned about performance; not just about the raw performance of their virtual machines, but mainly about the performance of the integration between the scripting engine and the embedding application. The applications I was working on needed fast throughput of streamed data; scripts were to be called hundreds (or thousands) times per second, each time with fresh new data coming from the outside. The transformation of this data into script items, back and forth, may have easily required more time than the script execution themselves.

Those concerns were simply not even considered in the design of all the mainstream embeddable languages, whose integrability in multithreading contexts ranged from difficult (causing slowdowns at application level due to excessive contention and context switches) to none, and whose item model ranged from unhandy (as representing all the numbers as floating point) to overly complex.

Only C/C++ in our modules

So I developed an engine that fit those needs. But while developing that engine, I realized that the same light and efficient interface to the application could work efficiently towards external modules. At that point I decided that all the standard modules for Falcon had to be written directly in C or C++ to ensure maximum performance to scripts.

Beyond an embeddable language

I always considered scripting languages as powerful tools to build even very complex applications, but only if those applications could count on powerful and extensive libraries of native code. We adopt a policy of zero scripting in libraries. While libraries of Falcon objects, classes and functions can be easily built by the final users, we will provide only native C/C++ modules, ensuring final Falcon applications the maximum speed they can get.

But the engine alone was not enough to justify a whole new language. Actually, I wasn't dissatisfied with the currently existing languages just because of their integration model or because of their VM. The language definitions themselves seemed too unorganized to a long-time C, C++ and Prolog developer. Some of the extreme liberties of untyped languages were just scratched, while others were too emphasized. Also, I didn't see the need to divide so deeply functional and procedural programming. I always believed that, except that for academic exercises, there was no need to "consider everything an object", or to disregard functional programming.

Mind over matter

I see programming languages as ways to let the mind act. The limits are in the fact that the listener of the thoughts is not able to understand them all, and needs a medium, a language, to be instructed. It's not the mind that must bend to formalization and modelling, for how powerful it may be, but it's the listener that should do its best to cope with the complexity of the thoughts of a human mind.

So, rather than adapting the thinking I had while programming to the tool I developed, I constantly adapted the tool to cope my thinking.

I have a relatively horizontal mind. I prefer knowing a bit of everything rather than everything of a bit. This is very probably the reason why I started to mix different programming styles, at first, and then whole paradigms, into Falcon.

However, the models that are currently available in Falcon, that is, procedural, object oriented,

prototype oriented, tabular, functional and message oriented programming are not just a list of different things. They are integrated into a single and unique blend of constructs, which orderly belong to one domain or the others, but that permeate and leverage the other domains to integrate them. For example, a function in a functional evaluated list may not just be a function; it may be a method of an existing instance as well:

```
// return a square of the argument
function sqr( a )
   return a*a
end

// create an object that extracts the (base)nth radix of an argument
// i.e.
// tenth_radix = radix_of( 10 )
// > tenth_radix.extract( exp(50) )   --> extracts the tenth radix of 50^e

class radix_of( base )
   _base = base

   function extract( value )
      return value ** (1/self._base)
   end
end

//create a functional "absolute" through the math sqr(x) operation
func_abs = .[ cascade .[      // this is the functional sequence operator
   sqr                        // our square argument
   radix_of( 2 ).extract      // get the extract method of an instance of square root.
   ]
]

// then call it
> func_abs( -4 )              // shall be 4
```

But also the reverse is possible, as functional constructs can be used as object members. For example:

```
// We'll use a singleton instance this time
object absoluter
   // a data to be filled by the functional operators
   _data = nil

   // and our functional operators...
   enact = .[cascade .[ self.set       // this will update self._data.
      self.square                      // and theese will use self._data
      self.sqrt] ]

   function set( value )
      self._data = value
   end

   function square()
      self._data *= self._data
   end

   function sqrt( value )
      return self._data ** 0.5
   end
end

// let's see it in action
> absoluter.enact( -2 )    // shall be 2
```

And this goes on, procedural programming becoming object oriented by setting members to normal functions, message programming becoming functional by broadcasting not just messages, but whole functions to be executed. Or to whole functional constructs based on methods...

Respect and the others

I shared my work because I wanted to share the benefits I received from the experience of creating Falcon to the other open source developers who had created the great products I had used.

But I do believe this language doesn't need an overall ruler to limit its possibilities. For however wise a ruler may be, he can never be good as an open group of experts integrating their experiences and working for a common goal.

This is the reason I dubbed the group of people working at Falcon the *Committee*, and the reason why I appointed myself with the role of "president". The people who joined our Committee have already experienced how their opinions, ideas, code and minds were appreciated, amplified, finalized and realized through the joint activity of all of us.

With no prejudices, with no prefigured limitations, our intellectual activity is all aimed to make the best possible scripting language ever, for it to be the best possible coding experience for our users.

What drives our activity is just **respect**, for the overall best technical solutions, for the users and for the community.

Giancarlo Niccolai

Elapsed time: 0.016 secs. (VM time 0.009 secs.)