Home

Ary Borenszweig edited this page Sep 7, 2016 \cdot 14 revisions

Welcome to the crystal wiki!

Website

• http://crystal-lang.org

Google Group

• Google Group

IRC

• channel #crystal-lang at irc.freenode.net

Editor support

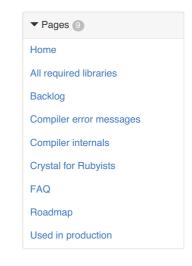
- Atom: language-crystal-actual and linter-crystal
- Emacs: emacs-crystal-mode
- Spacemacs: crystal-spacemacs-layer
- Sublime Text: sublime-crystal (named Crystal in Package Control)
- · Vim: vim-crystal
- Visual Studio Code: vscode-crystal and vscode-crystal-ide

Official Documentation

• http://crystal-lang.org/docs/ (in progress)

CI integrations

- Travis
- CircleCI



Clone this wiki locally

All required libraries

Roger Pack edited this page Jan 5, 2017 · 28 revisions

This is a list of known required libraries needed to run Crystal's specs.

General

You will need LLVM 3.5, 3.6, 3.8 or 3.9 (3.7 is not supported)

You will also need BOEHM GC 7.6 or greater (because of this commit). If your distro does not have a package, one way is to install the latest master (you may need autoreconf, automake, libtool, make, g++ and diff packages first):

```
git clone https://github.com/ivmai/bdwgc.git
cd bdwgc
git clone https://github.com/ivmai/libatomic_ops.git
autoreconf -vif
automake --add-missing
./configure
make
make check
sudo make install
```

To build crystal, a requisite is to have a working version of crystal itself already installed, see one of the non "from sources" option or use cross compilation to bootstrap a compiler for your current OS.

Ubuntu

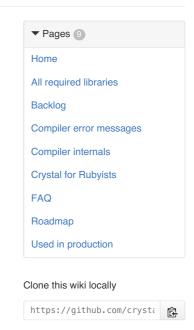
```
sudo apt-get install \
  libbsd-dev \
  libedit-dev \
  libevent-core-2.0-5 \
  libevent-dev \
  libevent-extra-2.0-5 \
  libevent-openssl-2.0-5 \
  libevent-pthreads-2.0-5 \
  libgmp-dev \
  libgmpxx4ldbl \
  libssl-dev \
  libxml2-dev \
  libyaml-dev \
  libreadline-dev \
  automake \
  libtool \
  git \
  llvm
```

You will likely have to install bdw gc from source, above.

Fedora

```
sudo dnf -y install \
  gmp-devel \
  libbsd-devel \
  libedit-devel \
  libevent-devel \
  libyaml2-devel \
  libyaml-devel \
  llvm-static \
  openssl-devel \
  readline-devel
```

Fedora 25 (current version as of writing) only packages Boehm GC 7.4. You can pull the 7.6 packages from Fedora Rawhide with:



```
sudo dnf install fedora-repos-rawhide
sudo dnf install gc gc-devel # get all dependencies from Fedora 25
sudo dnf install gc gc-devel --enablerepo=rawhide --best --allowerasing # upgrade only
```

This will not upgrade any other part of your system to Fedora Rawhide.

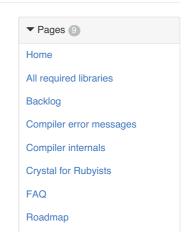
Mac OSX (with homebrew)

```
xcode-select --install
brew install \
   bdw-gc \
   gmp \
   libevent \
   libxml2 \
   libyaml \
   llvm
brew link llvm --force
```

Backlog

Ary Borenszweig edited this page Jul 6, 2014 \cdot 12 revisions

- Tipos genéricos y no genéricos
- Restricciones de tipos
- Self type en restricciones
- Código eficiente para uniones que tienen a Nil y sólo otros ObjectType
- Punteros, buffers, malloc, etc.
- Debugging
- REPL
- Bindings a C: pointers a structs, a buffers, out, etc.
- Quedarse con bloques, closures
- GC
- Alocar en el stack cuando sea posible
- No usar puntero a puntero si no es necesario
- Determinar si una variable de instancia es constante, para poder inlinearla
- Marcar funciones como void si su valor de retorno no se usa
- Return: en funciones que hacen yield y las cuales se invocan con un bloque
- Break
- Next
- Threads
- Primitivas de concurrencia
- Iteradores lazy
- Metaprogramación
- Fibers



Clone this wiki locally

Used in production



Compiler error messages

Stefan Merettig edited this page Nov 3, 2016 \cdot 2 revisions

can't infer block return type

For example:

```
class Foo
  def initialize(@name)
  end

def name
    @name
  end
end

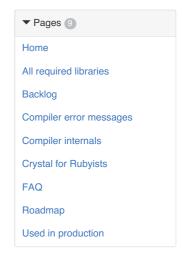
a = [] of Foo
x = a.map { |f| f.name } #=> error: can't infer block type, try to cast the block body
```

Here Foo was never instantiated so the compiler has no way of knowing what the type of @name is.

To solve this, cast the block body with as:

```
x = a.map { |f| f.name.as(String) } # works
```

In the future we want to get rid of this error messages and make the compiler smarter. In the above case <code>@name could</code> be deduced to be <code>Void or NoReturn</code>, but for now you have to use this workaround.



Clone this wiki locally

Compiler internals

yui-knk edited this page Jun 26, 2015 · 4 revisions

Here we explain how the compiler works. We link to the relevant code whenever possible, but since code changes we will use this version, which is a recent one and where later changes have a low impact on the general algorithm.

The main file

The main file that is compiled to generate the compiler issrc/compiler/crystal.cr. Here all source code relative to Crystal is required and then Crystal::Command.run is executed. The Command module provides a command line interface to the compiler. According to command line options, it creates a Compiler, configures it and then uses it to compile one or more source files.

Let's see what the Compiler class does.

The Compiler class

The main public method of the Compiler class iscompile.

First of all a Program is instantiated. A Program represents the top level container of everything. It's like a top level module that can have classes and methods. It's similar to Ruby's main when you do puts self at the top-level. However, unlike Ruby, when you define a method at the top level it gets defined in this Program, not as a private method of Object.

As you can see in Program's source code some basic types common to all programs are defined, like Object, Nil and String.

The Program is also a container of data associated to a single compilation, so for example it keeps track of all the symbols that were used (symbols can't be dynamically created), as well as some configurations, like CRYSTAL_PATH (similar to Ruby's \$LOAD_PATH, only it is immutable).

Going back to Compiler#compile, a Program is created and configured. Then the source code is parsed (also here). After parsing each file into an AST it is normalized. Normalization consists of transforming some AST nodes into others. The most important transformation is transforming a require into AST nodes that result from actually requiring that file. Other transformations are, for example, transforming an unless to an if by simply inverting the branches.

At the end of this stage we will have an AST node representing the whole program, with all requires expanded (the special "prelude" file is automatically required). This AST node will probably be an Expressions node, which just represents more than one AST node.

The next step is the most important one:type inference.

Type inference

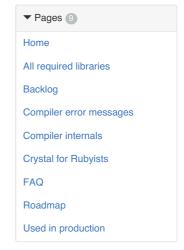
The name of this stage is actually misleading. It's called infer type in the source code, but many things happen here. This has two important consequences: 1) The code is harder to follow and understand, because many concerns are mixed, and 2) The compiler is faster, as the whole program is traversed just once. We believe point 2 is more important than 1, as there will be more developers using the compiler than developers developing the compiler, and compile times are very important for us.

Let's take a look at what Program#infer_type(node) does.

The first and most important thing it does is to create a TypeVisitor to traverse that AST node. This makes use of the Visitor Pattern, which is one that is heavily used across the compiler and it's one of the most useful ways to deal with an AST node in a generic way. Because of Crystal's multi-dispatch feature implementing the Visitor Pattern is very easy, as no manual double-dispatch is needed.

The TypeVisitor does many things:

- · It declares types and methods
- It binds AST nodes between each other to propagate type information



Clone this wiki locally



• It analyzes variable types and their flow

Type and methods declaration

When the user writes this:

```
class Foo
  class Bar
  def baz
    1
  end
  end
end
```

we want a Foo class to be declared (or reopened), a Bar class to be declared (or reopened) inside it, and a baz method inside it to be declared (or redefined).

For this, the TypeVisitor has a stack of types. The stack starts with a single element, @mod .

Note: throughout the code the word <code>mod</code> will be a synonym of the <code>Program</code>, as it is the global module that's always accessible. In some other cases <code>program</code> is used. The word <code>mod</code> comes from old times and we could change it now to <code>program</code>, only <code>mod</code> is very short and convenient.

Back to the stack of types, the stack starts with the Program. That means that when a type is defined, it will be defined in that type (the Program). Then when processing the class' body, this new type is pushed to the stack so new types will be defined underneath it. After processing the class' body the stack is popped. You can see this in action in visit(node: ClassDef). In that method there is a lot more code than just that, but it's mostly validations (for example: superclass mistmach, reopening a type as a module, namespace not found, etc), dealing with definition of generic types, running hooks (inherited).

A similar process happens in other definitions: Module, Enum, Lib, Alias, Include, Extend, Def and Macro.

AST nodes binding

To understand this section better we highly recommend you to read this blog post and this other one.

In short, the whole type inference algorithm is based on binding AST nodes to each other. When you bind a node A to B, A's type gets B's type. If B's type changes, A's type changes as well. If A is bound to another node C, A's type will be the union of B's type and C's type.

Every ASTNode has a bind_to method to bind itself to one or more nodes. When you bind a node A to a node B, B is added to A's dependencies and at the same timeB is added A as an observer. The result is that A's dependencies will be [B], and B's observers will be [A].

After a node is bound to others, its type is recomputed by doing atype merge of the dependencies' types. How types are merged is explained in the appendix. After the new type is computed it is propagated to suscribed observers by invoking their update method. The update method does or more or less the same thing: compute the new type based on the dependencies, only this information is not yet propagated. The node is marked as dirty and after all observers are updated propagate is invoked on them. This makes the propagation happen in small steps, preventing extra propagations.

Note: the above code is in the semantic/ast.cr file. There's also syntax/ast.cr, which defines the AST nodes and their properties. Everything under the semantic directory has something to do with the semantic stage, and it can (and does, a lot) reopen AST nodes to add more funcionality. This allows grouping funcionality in different files without having a huge ast.cr file with all the funcionality mixed.

Where is bind_to used? Let's see what happens when you write something like this:

```
a = 1
```

This is an Assign node, so the TypeVisitor will visit it. Since the target (the left-hand side) is a Var, this method will be invoked. First, the value is visited. In this case it's a NumberLiteral. Assigning a type to a NumberLiteral is easy: if it's an Int32 literal the type will be Int32. These types are well known, already defined (as we saw before) and accessed via the mod variable, which is the always-present Program. This is one of the few cases where bindings are not used. Other cases are Nil, Bool, Char and

other primitive types.

Going back to the type_assign method we can see that (amongst many other things)the target is effectively bound to the value. The node is also bound to the value because the Assign's node type is that of the value.

To be continued...

Crystal for Rubyists

David Kuo edited this page Jan 9, 2017 · 52 revisions

Although Crystal has a Ruby-like syntax, Crystal is a different language, not another Ruby implementation. For this reason, and mostly because it's a compiled, statically typed language, the language has some big differences when compared to Ruby.

Crystal as a compiled language

Using the crystal program

If you have a program foo.cr:

```
# Crystal
puts "Hello world"
```

When you execute one of these commands:

```
crystal foo.cr
ruby foo.cr
```

You will get this output:

```
Hello world
```

It looks like crystal interprets the file, but what actually happens is that the file foo.cr is first compiled to a temporary executable and then this executable is run. This behaviour is very useful in the development cycle as you normally compile a file and want to immediately execute it.

If you just want to compile it you can use the build command:

```
crystal build foo.cr
```

This will create a foo executable, which you can then run with ./foo .

Note that this creates an executable that is not optimized. To optimize it, pass the --release flag:

```
crystal build foo.cr --release
```

When writing benchmarks or testing performance, always remember to compile in release mode.

You can check other commands and flags by invoking crystal without arguments, or crystal with a command and no arguments (for example crystal build will list all flags that can be used with that command).

Types

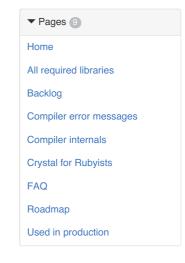
Bool

true and false are values in the Bool class rather than values in classes True Class or False Class.

Integers

For Ruby's Fixnum type, use one of Crystal's Integer types Int8, Int16, Int32, Int64, UInt8, UInt16, UInt32, or UInt64.

If any operation on a Ruby Fixnum exceeds its range, the value is automatically converted to a Bignum. Crystal will use modular arithmatic on overflow. For example:



Clone this wiki locally

```
x = 127_i8  # An Int8 type
puts x # 127
x += 1 # -128
x += 1 # -127
```

See Integers

Regex

```
Global variables \ and \ are missing (yet \ and \ , \ , ... are present). Use \ -pre_match and \ -post_match read more
```

Paired-down instance methods

In Ruby where there are several methods for doing the same thing, in Crystal there may be only one. Specifically:

```
Ruby Method Crystal Method
-----
Enumerable#detect Enumerable#find
Enumerable#collect Enumerable#map
Object#respond_to? Object#responds_to?
length, size, count
```

Omitted Language Constructs

Where Ruby has a a couple of alternative constructs, Crystal has one.

- trailing while/until. Note however that if as a suffix is still available
- and and or : use && and || instead with suitable parenthesis to indicate precedence
- Ruby has Kernel#proc , Kernel#lambda , Proc#new and -> , while Crystal uses just ->
- For require_relative "foo" use require "./foo"

No autosplat for arrays and enforced maximum block arity

```
[[1, "A"], [2, "B"]].each do |a, b|
pp a
pp b
end
```

will generate an error message like

```
in line 1: too many block arguments (given 2, expected maximum 1)
```

However omitting unneeded arguments is fine.

There is autosplat for tuples:

```
[{1, "A"}, {2, "B"}].each do |a, b|
pp a
pp b
end
```

will return the result you expect.

Reflection and Dynamic Evaluation

Kernel#eval() and the weird Kernel#autoload() are omitted. Object and class introspection methods Object#kind_of?(), Object#methods, Object#instance..., and Class#constants, are omitted.

In some cases macros can be used for reflection.

Semantic differences

single- versus double-quoted strings

In Ruby, string literals can be delimited with single or double quotes. A double-quoted string in Ruby is subject to variable interpolation inside the literal, while a single-quoted string is not.

In Crystal, strings literals are delimited with double quotes only. Single quotes act as character literals the same as say C-like languages. As with Ruby, there is variable interpolation inside string literals.

In sum:

```
X = "ho"
puts '"cute"' # Not valid in crystal, use "\"cute\"", %{"cute"}, or %("cute")
puts "Interpolate #{X}" # works the same in Ruby and Crystal.
```

Triple quoted strings literals of Ruby or Python are not supported, but string literals can have newlines embedded in them:

```
"""Now,
what?""" # Invalid Crystal use:
"Now,
what?" # Valid Crystal
```

```
The [] and []? methods
```

In Ruby the [] method generally returns nil if an element by that index/key is not found. For example:

```
# Ruby
a = [1, 2, 3]
a[10] #=> nil

h = {a: 1}
h[1] #=> nil
```

In Crystal an exception is thrown in those cases:

```
# Crystal
a = [1, 2, 3]
a[10] #=> raises IndexOutOfBounds

h = {a: 1}
h[1] #=> raises MissingKey
```

The reason behind this change is that it would be very annoying to program in this way if every Array or Hash access could return <code>nil</code> as a potential value. This wouldn't work:

```
# Crystal
a = [1, 2, 3]
a[0] + a[1] #=> Error: undefined method `+` for Nil
```

If you do want to get nil if the index/key is not found, you can use the []? method:

```
# Crystal
a = [1, 2, 3]
value = a[4]? #=> return a value of type Int32 | Nil
if value
  puts "The number at index 4 is : #{value}"
else
  puts "No number at index 4"
end
```

The []? is just a regular method that you can (and should) define for a container-like class.

Another thing to know is that when you do this:

```
# Crystal
h = {1 => 2}
h[3] ||= 4
```

the program is actually translated to this:

```
# Crystal
h = {1 => 2}
h[3]? || (h[3] = 4)
```

That is, the []? method is used to check for the presence of an index/key.

Just as [] doesn't return nil, some Array and Hash methods also don't return nil and raise an exception if the element is not found: first, last, shift, pop, etc. For these a question-method is also provided to get the nil behaviour: first?, last?, shift?, pop?, etc.

The convention is for obj [key] to return a value or else raise if key is missing (the definition of "missing" depends on the type of obj) and for obj [key]? to return a value or else nil if key is missing.

For other methods, it depends. If there's a method named foo and another foo? for the same type, it means that foo will raise on some condition while foo? will return nil in that same condition. If there's just the foo? variant but no foo, it returns a truthy or falsey value (not necessarily true or false).

Examples for all of the above:

- Array#[](index) raises on out of bounds, Array#[]?(index) returns nil in that case.
- Hash#[](key) raises if the key is not in the hash, Hash#[]?(key) returns nil in that case.
- Array#first raises if the Array is empty (there's no "first", so "first" is missing), while
 Array#first? returns nil in that case. Same goes for pop/pop?, shift/shift?, last/last?
- There's String#includes?(obj), Enumerable#includes?(obj) and Enumerable#all?, all of which don't have a non-question variant. The previous methods do indeed return true or false, but that is not a necessary condition.

for loops

for loops are currently missing but you can add them via macro:

```
macro for(expr)
  {{expr.args.first.args.first}}.each do |{{expr.name.id}}|
     {{expr.args.first.block.body}}
    end
end

for i in [1,2,3] do
    puts i
end
# note the trailing 'do' as block-opener!
```

Properties

The ruby <code>attr_accessor</code> , <code>attr_getter</code> and <code>attr_setter</code> methods are replaced with new keywords:

```
Ruby Keyword Crystal Keyword
------
attr_accessor property
attr_reader getter
attr_writer setter
```

And && or II

Nice english operators for '&&' and 'II' are currently not supported

Verbose brackets()

In general you need some more brackets to compile

```
def brackets_needed(a)
  a.is_a?(Array)
end
```

Consistent dot notation

Ruby File::exists? becomes crystal File.exists? etc...

Crystal keywords

Crystal added some new keywords, these can still be used as function names, but need to be called explicitly with dot: e.g. $self.select{ |x| x > "good" }$

Why isn't there Windows support?

Windows support will eventually come. The reasons it's not currently supported are:

- 1. Windows APIs are different than linux/mac, which are mostly POSIX-compliant.
- 2. None of the core developers use Windows so there's no "dog-fooding" need for it. Core developers use mac/linux, either as desktop machines or servers.
- 3. Travis doesn't support Windows, so even if we add basic support for it, if the language and standard library continue evolving and we don't have a reliable way to test that Windows support doesn't break then it's not of much use.

We repeat: Windows support will definitely come in the future, but right now it's more likely to come in the form of a PR contributed to the project.

Why isn't the language indentation based?

Apart from the "Crystal has Ruby-inspired syntax" reason, there are more reasons:

- 1. If you copy and paste a snippet of code, you have to manually re-indent the code for it to work. This slows you down if you just wanted to do a quick test. And, since Crystal has a built-in formatter, it can re-indent the code automatically for you.
- 2. If you want to comment some code, for example comment an if condition, you have to re-indent its body. Later you want to uncomment the if and you'll need to re-indent the body. This slows you down and it's cumbersome.
- 3. Macros become harder to write. Consider the json_mapping macro. It defines def s, uses case ... when ... else ... end without having to bother whether the generated code will be indented. Without end , the user would have to correctly indent the lines that would be generated.
- 4. If you want a template language like ERB or ECR for a language that doesn't care about whitespace, you'll have to put those end to signal where conditions/loops/blocks end.
- 5. Right now you can do: [1, 2, 3].select { |x| x.even? }.map { |x| x.to_s }. Or you can do it with do .. end . How would you chain calls in an indentation-based language? Usage of { ... } is not valid, only indentation should be used to match code blocks.
- 6. Assuming one day we have a REPL, in which you tend to write code quickly, it's tedious and bugprone to match indentation, because whitespace is basically invisible.

Because of all the above reasons, know that the end keyword is here forever: there's no point in trying to suggest changing the language to an indentation-based one.

Why don't you add syntax for XYZ?

Before suggesting syntax additions, ask this question:

• Can it be currently done with the current syntax?

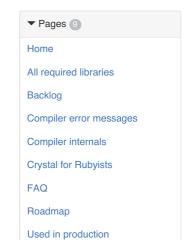
If the answer is "yes", there's probably no need to add new syntax for something that can be already done. Adding syntax means we have to be sure it doesn't conflict with the existing syntax. all users will have to learn something new, and it needs to be documented.

Maybe the current syntax is long to write or involves a couple of composed methods, but we should favor method composition instead of specific rules for specific problems.

Language X has feature Y. Why don't you have such feature?

If language X is not similar to Crystal (for example, language X has no mutable data, or is purely functional) then chances are that feature Y exists in language X because without it programming would be tedious or maybe impossible. In this case chances of adding Y to Crystal are null.

Some examples:



Clone this wiki locally

- Making the GC optional: impossible, the whole language needs to change.
- Making all data immutable by default: impossible, the whole language and standard library needs to change.
- Adding Elixir's pipe operator (|>): in OOP languages we have methods and self . See #1388.
- Adding Elixir/Erlang guards: not really needed, use if , is_a? or type restrictions.

Please don't insist on these things because they will never, ever happen.

Why trailing while/until is not supported, unlike Ruby?

In Ruby a trailing while comes in two flavors:

We find this logic confusing, and even Matz regrets it.

We had four options:

- 1. Keep the same semantic as Ruby.
- 2. Unify the semantic of both constructs.
- 3. Disallow the second construct (which Matz seems to regret)
- 4. Disallow both constructs.

We didn't want Option 1 because that would be to keep a mistake.

Option 2 is the worst choice because it will be surprising for those who come from Ruby: their code will compile fine but behave in a different way.

Option 3 sounds good, but for someone learning Crystal without previous Ruby experience, we think it might be confusing. Imagine you don't know Ruby's semantic and you see this:

```
<code> if <condition>
```

Here, it doesn't make sense to execute <code> without first checking <condition> . However, if we change the if to a while :

```
<code> while <condition>
```

there are now two possibilities: execute <code> and then check the <condition> (in a loop), or check the <condition> and then execute <code> (in a loop). And <code> comes before <condition> , so you might consider that possibility.

So, to remove all ambiguity, so that programmers don't have to stop thinking about what happens first, we decided to go with Option 4.

You can always replace this:

```
# Ruby
<code> while <condition>
```

with this:

```
# Crystal and Ruby
while <condition>
  <code>
end
```

and this:

```
# Ruby
begin
     <code>
end while <condition>
```

with this:

while is used much less frequently than if, so we think this is the correct choice.

As a bonus, the compiler's code and logic becomes simpler, because there's only one mode of operation for while, and there are less things to learn.

Why are parentheses mandatory for def arguments?

The main reason is that this:

```
def method arg : String
end
```

is ambiguous: is String an argument type restriction, or the method's return type? Even if we always associate to the left, it's confusing because one usually scans past the last colon to check the return type, and here you can't do that.

The second reason is that it makes def s have a single, unified style across a project, and between projects. In our experience leaving the parentheses off leads to some discussions between members of a project. These discussions disappear if there is only one way to do it.

Why are aliases discouraged?

Ruby has many aliases: length , size and count for Array, Hash, String and Enumerable. There's also map / collect , find / detect , select / find_all , etc. In our opinion, this is bad:

- Having more than one way to do a single thing implies learning more: you have to know all of the aliases to potentially understand code, because someone else might use an alias you don't use.
- If you want to implement a type similar to Array, you have to define all of the aliases for someone's code to work. In Crystal, where implementing efficient containers is possible (in Ruby too, but you probably have to do it in C), this is very important to make it easy to do this.
- In a dynamically typed language, that alias definition must exist in memory for no real reason. In a
 statically typed language that alias must exist somewhere, slowing down (a bit) the semantic and
 codegen phase, and ending with a (slightly) bigger executable.
- It opens up the door for useless discussion: should length be preferred over count ?

Roadmap

Ary Borenszweig edited this page Feb 20, 2016 · 17 revisions

Roadmap

This roadmap defines the things that we definitely want to have in the language and plan to do. It can grow over time, but it can only shrink once we do the tasks.

Language

Stuff that has to do with the language syntax, semantic and runtime.

Concurrency support

Define how to do channel select			
Fix/check	<pre>I0.select</pre>		
	Process.run	(waitpid must not l	olock)
Run fibers on multiple threads, with a single IO loop and job stealing			
Add concurrency primitives like WorkGroup			
∑ For 32	2 bits		
∑ For 64	4 bits		

Handle stack overflows

Stack overflows should be detected and shown (https://github.com/manastech/crystal/issues/271)

Process execution

Decide how to execute external processes (Process.run). There are already three different pull requests about this.

Platform

Stuff that has to do with where and how Crystal runs.

32-bits support

- Fix wrong assumptions about C types
- Fix broken specs (BigInt, etc.)
- Add to omnibus
- Add Vagrant config
- Find out a way to automatically run specs for 32 bits (travis doesn't support it)

Tools

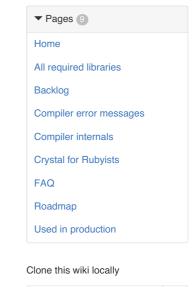
Built-in tools integrated to the crystal command.

Package manager

Crystal will provide a built-in package manager. We really want this to be theonly package manager so it's easier to build a collaborative community.

We want a truly decentralized package manager. These ideas could make it work:

- No global directory where all deps are installed: it's local to each project (lib/libs directory)
- Each repo has a special branch (maybe _releases) with metadata for dependencies for each version. These are cummulative, so version 0.2 contains metadata for 0.1 and 0.2.
- Crystal provides a command to release a new version, pushing to that special branch, and creating a tag



- Use Semantic Versioning
- · Automatically download recursive dependencies
- · Resolve conflicts
- Remove Projecfile , use YAML, both for project.yml and for the metadata file
- Include name and version in project.yml
- The name above is used for the directory in which it is installed, and used by require "...", so for example "webmock.cr" will be installed in "lib/webmock", and that directory will contain the direct checkout of the project (so it has the src directory in it)
- The require logic changes to that if you do require "foo", we check if there's foo.cr in CRYSTAL_PATH, or foo/foo.cr, or foo/src/foo.cr (this last one is the one that will be used for dependencies installed via the package manager). With this logic, the current CRYSTAL_PATH value doesn't need to change
- Probably rename "libs" to "lib"

With the above, when you do crystal deps install, all first-level dependencies are gathered. From there we go to each depednency's repository and check out the special _releases branch to get all metadata for all versions of that first-level dependency. We apply this recursively until we get all the metadata for all involved libraries. The previous process should be fast, because only that metadata branch must be checked out, and only once for each library (and we can parallelize the requests). Then we can solve conflicts and install what's needed.

For discoverability, we can list github/bitbucket repositories that have crystal code and that also have that special _releases branch, which in turn contains all the information for every version of the library.

For all of this, the easiest thing would be to build on top of @ysbaddaden'sshards, which already has the desired YAML format, probably has some logic for semver, etc.

Automatic build/run

It would be awesome if you could download a project and just do crystal build or crystal run without arguments, and that builds/runs the default executable.

For this, we can use the name of the project (specified in project.yml) and use $./src/{name}/{name}$.

This argless version also executes <code>crystal deps</code> , so that doing <code>crystal build/run</code> works out of the box (given you have the necessary dependencies).

Docs generator

Nicer style Nicer
Show inherited methods from superclasses and included modules
□ Don't use HTML frames because they are deprecated/discouraged (optional)
Support inter-linking between docs (we can start withhttp://www.docrystal.org/)
$\ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ $
for the standard library)

Debugger

This section needs to be defined.

Standard library

Stuff provided by the standard library. If you wish to tackle some of these, please let us know! But first open an issue so we can discuss the best way to do it (and we'll link the issue in this page.)

Some types listed here might already exist but have incomplete functionality, or must be reviewed (we might need to mark this in the code somehow, similar to how, for example, Rust does it.)

Used in production

benoist edited this page Dec 24, 2016 \cdot 8 revisions

Crystal is still changing and growing rapidly. Here we list the brave folks that start to use it in production nonetheless, building and strengthening our community. Are you using Crystal in production at your company or project? Please add yourself to the list under the corresponding industry!

Gaming

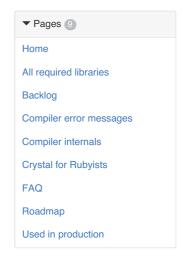
• Neopoly GmbH - Online games and services.

SaaS products

· Appmonit - analytics engine

Software Development

- Manas We build unconventional software for unconventional [needs I contexts I ideas I organizations]
- Protel Changing the game in POS Systems and Hospitality.
- Bulutfon Cloud Voice & VOIP solution for Turkey.



Clone this wiki locally

