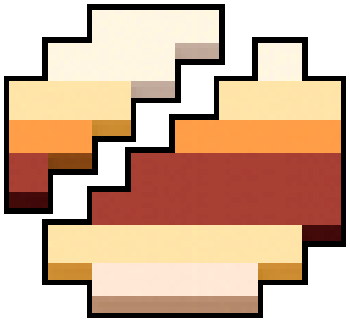

JOdin

API

github.com/nik1oo/jodin

---

jodin 0.1.0-alpha MIT License

Contents

Things we steal from IPython

Comprehensive object introspection

You can write `jodin.inspect(x)` to print details about object `x`. You can use the format verbs for this:

- `%#v` — the value in a default format
- `%w` — an Odin-syntax representation of the value
- `%T` — an Odin-syntax representation of the type of the value

Should look like this:

```
Value: <value>
Type:  <type>
Size:  <size>
Doc:   <docstring from the odin source code>
```

Followed by more fields unique to the particular type.

Input history, persistent across sessions

Extensible system of ‘magic’ commands for controlling the environment and performing many tasks related to IPython or the operating system.

A rich configuration system with easy switching between different setups (simpler than changing `$PYTHONSTARTUP` environment variables every time).

Session logging and reloading.

Access to the system shell with user-extensible alias system.

Create a rich display of Html, Images, Latex, Sound and Video.

interactive widgets with the use of the `ipywidgets` package.

Searching through the installed packages

The `%psearch` magic command from IPython:

Writing `jodin.psearch("builder")` would execute `grep -r "builder" /c/Program\ Files/Odin/` in the system shell and print the results.

System Shell Access

The equivalent to `var = !cmd` from IPython:

`jodin.shell("<command>")` is a powered-up version of `libc.system("<command>")`. It executes a command in the system shell and captures the output in a string.

Filesystem Navigation

The equivalent to `%cd` from IPython:

You can change `__dir__` by `jodin.cd()`. You can list the contents of `__dir__` by `jodin.ls()`.

Flexible configuration system

`jodin profile create` to create a new configuration profile. It goes into `~/.jodin/profile_default`.

Simple timing information

The equivalent to `%timeit` from IPython:

Either `#+timeit` (for the entire cell) or `#timeit` (for the next expression) uses `time.Stopwatch` to track the execution time of the cell and print it at the end of its execution.

Help system

`#jodin.help()` prints help information.

Magic commands

`%` applies to line. `%%` applies to cell.

Functions that work with code: `%run`, `%edit`, `%save`, `%macro`, `%recall`, etc. Functions which affect the shell: `%colors`, `%xmode`, `%automagic`, etc. Other functions such as `%reset`, `%timeit`, `%%writefile`, `%load`, or `%paste`.

Running

Call `%run` with a path to an odin package with a `main` proc and this package will be executed as if it was copied to that cell and `main` was `__main__`.

1. Code Generation Rules

The content of each cell is parsed by `"core:odin/parser"` into an Odin AST. The source of each cell is generated from the AST by the following rules:

A. A main procedure is generated.

B. The declarations generated inside the main procedure are in the same order as the corresponding declarations from which they were generated.

C. Every root-level declaration of type `Value_Decl` that has field `is_mutable` set to `true` and that has values generates an equivalent `Value_Decl` without values in root scope and an equivalent `Assign_Stmt` declaration in main scope.

D. Every root-level declaration of type `Value_Decl` with field `is_mutable` set to `false` generates an identical declaration in root scope.

2. Linking Rules

To enable cells to share symbols and data, the following rules are applied:

A. For each `<x> : T` where `! intrinsics.type_is_pointer(T)`, assignment is allowed in the declaration cell and in export cells.

B. For each `<x> := T` where `intrinsics.type_is_pointer(T)`, dereferenced assignment and pointer assignment are allowed in the declaration cell and in export cells, and the preprocessor must insert `__update_symmap__()` after every pointer assignment.

C. A global declaration of symbol `<x>` in the content of a cell is an error, if a global declaration of `<x>` already exists in the content of a previously-compiled cell, unless one of them is given the attribute `@(export=false)`.

D. Every global declaration of symbol `<x>` in the source code of a cell is given the attribute `@(export)`, unless `<x>` was given the attribute `@(export=false)` in the content of the cell where it was originally declared.

Every variable `x` of type `T` has one *declaration cell* and zero or more *export cells*. The value of `x` must be accessible by the same symbol `x` from the declaration cell as well as every export cell.

The following rules apply to every variable declaration `x: T` that is to be placed in root scope:

(2.1):
(2.1): If `intrinsics.type_is_pointer(T)`, `x` is declared in export cells as `x: T`.
(2.2): Otherwise, `x` is declared in export cells as `x: ^T`.

3. Memory Rules

To enable the preservation of memory across compilation and recompilation of cells, the following rules are applied:

4. Package API

The JOdin package should be located in `<Odin>/shared/jodin`, which is implicitly imported in every cell. Everything is under namespace `jodin`.

Context

```
Context :: struct {
  pretty_print: (object: []u8, type_info: ^runtime.Type_Info) }
```

Object holding information about the current cell.

Cell_Info

```
Cell_Info :: struct {
  id:   string,
  name: string,
  code: string }
```

Object holding information about the current cell.

Audio_Format

```
Audio_Format:: enum u8 {
  AAC,
  MP3,
  WAV,
  WEBM }
```

Audio formats supported by the Jupyter front-end.

Image_Format

```
Image_Format:: enum u8 {
  PNG,
  JPEG,
  GIF,
  WEBP }
```

Image formats supported by the Jupyter front-end.

cell_info

```
cell_info:: proc() -> Cell_Info {…}
```

Get info about the current cell.

stream

```
stream:: proc(which: enum { STDOUT, STDERR }, text: string) -> bool {…}
```

Write a string of text to the STDOUT/STDERR stream of the front-end.

display_data

```
display_data:: proc(data: []u8, mime_type: string, width: uint = 0, height: uint = 0,
expanded: bool = true, display_id: string) -> bool {…}
```

Display data in the front-end.

display_audio

```
display_audio:: proc(data: []u8, format: Audio_Format, element_id: string = "") -> bool
{…}
```

Play an AAC/MP3/WAV/WEBM audio in the front-end.

display_image

```
display_image:: proc(data: []u8, format: Image_Format, width: uint, height: uint,
display_id: string = "") -> bool {…}
```

Display a PNG/JPEG/GIF/WEBP image in the frontend.

clear_output

```
clear_output:: proc(wait: bool = false) -> bool {…}
```

Clear the output currently visible in the front-end. If `wait` is `true`, the output will be cleared only when new output becomes available.

active_types

```
active_types:: proc() -> []string {…}
```

List the MIME types implemented by the display formatted of the current active front-end.

format

```
format:: proc(value: $T) -> map[string]Message {…}
```

List the MIME types implemented by the display formatted of the current active front-end.