Indium Documentation

Release 2.1.1

Nicolas Petton

Jul 15, 2019

Contents

Indium is a JavaScript development environment for Emacs.

Indium is Free Software, licensed under the GPL v3.0. You can follow its development on GitHub.

Indium connects to a browser tab or nodejs process and provides several features for JavaScript development, including:

- a REPL (with auto completion) & object inspection;
- an inspector, with history and navigation;
- a scratch buffer (`M-x indium-scratch`);
- JavaScript evaluation in JS buffers with `indium-interaction-mode`;
- a stepping Debugger, similar to `edebug`, or `cider`.

This documentation can be read online at https://indium.readthedocs.io and in Info format (within Emacs with `(info "Indium")`).

It is also available in Info format and can be consulted from within Emacs with `C-h i m indium RET`.

CHAPTER 1

Table of contents

1.1 Installation

Note: If you already have installed `Jade`, you should read the migration-from-jade page first.

Indium supports `Emacs 25.3+`, works with `Chrome` (debugging protocol `v1.2`, see *Chrome/Chromium requirements*) and `NodeJS`, see *NodeJS requirements*.

Indium works with `js-mode`, `js2-mode`, `js2-jsx-mode` and `rjsx-mode`. It supports the ECMAScript features of the runtime it connects to.

Indium is availabe on MELPA, MELPA Stable.

1.1.1 The Indium server

Indium needs to communicate with a small server for evaluation and debugging. Install the server with the following command (prepend `sudo` on GNU/Linux):

```
npm install -g indium
```

1.1.2 Using MELPA

Unless you are already using MELPA, you will have to setup `package.el` to use MELPA or MELPA Stable repositories. You can follow this documentation.

You can install Indium with the following command:

```
M-x package-install [RET] indium [RET]
```

or by adding this bit of Emacs Lisp code to your Emacs initialization file (`.emacs` or `init.el`):

```
(unless (package-installed-p 'indium)
  (package-install 'indium))
```

If the installation doesn't work try refreshing the package list:

```
M-x package-refresh-contents [RET]
```

1.1.3 Manual installation

If you want to install Indium manually, make sure to install `websocket.el`. Obtain the code of Indium from the repository.

Add the following to your Emacs configuration:

```
;; load Indium from its source code
(add-to-list 'load-path "~/projects/indium")
(require 'indium)
```

1.2 Getting up and running

1.2.1 Project configuration

Place a `.indium.json` file in the root folder of your JavaScript project. The project file can contain one or many configurations settings for NodeJS (see *NodeJS configuration options*) and Chrome/Chromium (see *Chrome/Chromium configuration options*).

Here is a minimalist `.indium.json` file.:

```
{
  "configurations": [
    {
      "name": "Web project",
      "type": "chrome"
    }
  ]
}
```

1.2.2 General configuration

The `.indium.json` file can contain as many configurations as needed, and mix any supported configuration types.

The currently supported `type` values are `"chrome"` and `"node"`.

The root directory of the source files is by default set to the directory where this `.indium.json` file is placed, but it can be overridden with the `root` (or the `webRoot` alias) option:

```
{
  "configurations": [
    {
      "type": "chrome",
      "root": "src"
    }
```

(continues on next page)

(continued from previous page)

```
  ]
}
```

Custom script path overrides can be set with `scriptPathRegexpOverrides`. See *Overriding script paths* for more information on script paths and debugging.

Custom sourcemap path overrides can be set with `sourceMapPathOverrides`. See *Using sourcemaps* for more information on sourcemaps and debugging.

1.2.3 Chrome/Chromium configuration options

host Host on which Chrome is running (defaults to `"localhost"`).

port Port on which Chrome is running (defaults to `9222`).

url Url to open when running `indium-launch`.

Example configuration:

```
{
  "configurations": [
    {
      "name": "Web project",
      "type": "chrome",
      "host": 192.168.22.1,
      "url": "http://192.168.22.1/myproject/index.html",
      "port": 9222
    }
  ]
}
```

1.2.4 NodeJS configuration options

command Nodejs command to start a new process. The `--inspect` flag will be added automatically.

inspect-brk Whether Indium should break at the first statement (false by default).

host Host on which the Node inspector is listening (defaults to `"localhost"`).

port Port on which the Node inspector is listening (defaults to 9229).

Here is an example configuration for debugging Gulp tasks:

```
{
  "configurations": [
    {
      "name": "Gulp",
      "type": "node",
      "command": "node ./node_modules/gulp/bin/gulp.js",
      "inspect-brk": true
    }
  ]
}
```

1.2.5 Starting Indium

Indium can be started in two modes:

- Connect: `M-x indium-connect` Connect indium to a running runtime from one of the configurations in the `.indium.json` project file.
- Launch: `M-x indium-launch` Start a JavaScript process (Chrome or NodeJS) as specified from the configurations in the `.indium.json` project file.

1.2.6 NodeJS requirements

Nodejs >= `8.x` is required for Indium to work.

If your distribution ships an old version of NodeJS, you can install a more recent version using nvm:

```
$ curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.33.2/install.sh | bash
```

Once `nvm` is install, you can easily install and use the version of NodeJS you want:

```
$ nvm install v8
$ nvm alias default v8
$ node --version
```

If you install NodeJS using `nvm`, chances are that Emacs won't have it in its `exec path`. A simple solution is to use the excellent exec-path-from-shell package.

1.2.7 Chrome/Chromium requirements

Chrome/Chromium >= `60.0` is required for Indium to properly work (debugging protocol `v1.2`).

When the variable `indium-chrome-use-temporary-profile` is non-nil (the default), `M-x indium-launch` will start a new instance of Chrome/Chromium with the remote debugging port set up.

Otherwise, you can start Chrome/Chromium with the `--remote-debugging-port` flag like the following:

```
chromium --remote-debugging-port=9222 https://localhost:3000
```

If you start Chrome manually, make sure that no instance of Chrome is already running, otherwise Chrome will simply open a new tab on the existing Chrome instance, and the `remote-debugging-port` will not be set.

1.3 The REPL

1.3.1 Starting a REPL

A REPL (Read Eval Print Loop) buffer is automatically open when a new Indium connection is made (see *Getting up and running*).

The REPL offers the following features:

- Auto completion with `company-mode`
- JS syntax highlighting
- Pretty printing and preview of printed values
- Access to the object inspector (see *The inspector*)

1.3.2 Using the REPL

Keybindings

Here is the list of available keybindings in a REPL buffer:

Keybinding	Description
`RET`	Evalute the current input. When the point is on a printed object, inspect the object.
`C-RET`	Insert a newline.
`C-c M-i`	Evalute the current input and open an inspector on the result.
`C-c C-o`	Clear the output.
`C-c C-q`	Kill the REPL buffer and close the current connection.
`M-n`	Insert the previous input in the history.
`M-p`	Instert the next input in the history.

Reconnecting from the REPL buffer

When a connection is closed (most probably because other devtools were open on the same runtime), the REPL will display two buttons, one to try to reopen the connection, and another one to kill Emacs buffers using this connection (the REPL buffer, inspectors & debuggers).

1.3.3 Code evaluation & context

When evaluating code in the REPL, Indium will always run the code on the current execution context.

This means that while debugging, code execution will happen in the context of the current stack frame, and will be able to access local variables from the stack, etc.

1.4 Interaction in JS buffers

Indium comes with a minor mode called `indium-interaction-mode` for interactive programming. To enable it in all JavaScript buffers, add something like the following to your Emacs configuration:

```
(require 'indium)
(add-hook 'js-mode-hook #'indium-interaction-mode)
```

When `indium-interaction-mode` is on, you can evaluate code, inspect objects and add or remove breakpoints from your buffers.

1.4.1 Evaluating and inspecting

Here's a list of available keybindings:

- `C-x C-e`: Evaluate the JavaScript expression preceding the point.
- `C-M-x`: Evaluate the innermost function enclosing the point.
- `C-c M-i`: Inspect the result of the evaluation of an expression (see *The inspector*).
- `C-c M-:`: Prompt for an expression to evaluate and inspect.
- `M-x indium-eval-buffer`: Evaluate the entire buffer.
- `M-x indium-eval-region`: Evaluate the current region.

1.4.2 Switching to the REPL buffer

Press `C-c C-z` from any buffer with `indium-interaction-mode` turned on to switch back to the REPL buffer (see *The REPL*).

1.4.3 Adding and removing breakpoints

You need to first make sure that Indium is set up correctly to use local files (see *General configuration*).

- `C-c b b`: Add a breakpoint
- `C-c b c`: Add a conditional breakpoint
- `C-c b k`: Remove a breakpoint

- `C-c b t`: Toggle a breakpoint
- `C-c b K`: Remove all breakpoints from the current buffer
- `C-c b e`: Edit condition of a breakpoint
- `C-c b l`: List all breakpoints and easily jump to any breakpoint
- `C-c b d`: Deactivate all breakpoints (the runtime won't pause when hitting a breakpoint)
- `C-c b a`: Activate all breakpoints (it has no effect if breakpoints have not been deactivated)

The left fringe or margin can also be clicked to add or remove breakpoints.

Once a breakpoint is set, execution will stop when a breakpoint is hit, and the Indium debugger pops up (see *The stepping debugger*).

Since Indium 0.7, breakpoints are supported in source files with an associated sourcemap, see *Using sourcemaps*.

Note: Breakpoints are persistent: if the connection is closed, when a new connection is made Indium will attempt to add back all breakpoints.

1.5 The stepping debugger

1.5.1 Using sourcemaps

Since version 0.7, Indium uses sourcemap files by default.

For sourcemaps to work properly with Chrome/Chromium, make sure that a workspace is correctly set (see *Getting up and running*).

Warning: If your project uses sourcemaps, we advise you to use `js-mode` with `js2-minor-mode` instead of `js2-mode`. `js2-mode` can be extremely slow at parsing large files (like compiled JavaScript files) that the debugger might open if a stack frame source is not source-mapped. This can happen for instance when using Webpack.

Overriding sourcemap paths

Some sourcemaps cannot be used as is and need path rewriting to map to locations on disks.

Indium provides the configuration option `sourceMapPathOverrides` for providing custom sourcemap paths.

The default mapping works well for Webpack projects:

```
{
  "webpack:///./~/": "${root}/node_modules/",
  "webpack:///./":   "${root}/",
  "webpack:///":     "/",
  "webpack:///src/": "${root}/"
}
```

Overriding the `sourceMapPathOverrides` option will erase the default mapping.

Tip: If sourcemaps do not seem to work, you can see how Indium resolves sourcemap paths using `M-x indium-list-sourcemap-sources`.

Overriding script paths

If your application's script URLs don't correspond directly to where their source code is located, you can use `scriptPathRegexpOverrides` to tell Indium where to find the sources. It maps regular expressions to Javascript substitution strings.

For example, if your project root is `/home/user/projects/foo/`, and the source code for http://localhost:3000/js/app.js/1234567890 is at `/home/user/projects/foo/private/js/app.js`, you might set `scriptPathRegexpOverrides` to:

```
{
  "(/js/.*\\.js)/[0-9]+": "private$1"
}
```

This removes the trailing slash and digits, and it adds "private" to the beginning of the path below the project root.

1.5.2 Blackboxing scripts

The custom variable `indium-debugger-blackbox-regexps` holds a list of regular expression of script paths to blackbox when debugging.

Blackboxed scripts are skipped when stepping in the debugger.

1.6 The inspector

Indium features an object inspector that can be open on any object reference from a REPL buffer (see *The REPL*), the debugger (see *The stepping debugger*), or the result of any evaluation of JavaScript code (see *Interaction in JS buffers*).

To inspect the result of the evaluation of an expression, press `C-c M-i`. An inspector buffer will pop up. You can also press `RET` or left click on object links from the REPL buffer.

1.6.1 Using the inspector

Here is a list of available keybindings in an inspector buffer:

Keybinding	Description
`RET`	Follow a link to inspect the object at point
`l`	Pop to the previous inspected object
`g`	Update the inspector buffer
`n` or `TAB`	Jump to the next object in the inspector
`p` or `s-TAB`	Jump to the previous object in the inspector

1.7 Troubleshooting

If you run into issues with Indium, this document might help you.

1.7.1 General advice before reporting issues

Issues should be reported on the GitHub issue tracker.

1. If you encounter errors, you can enable `debug-on-error` in Emacs using `M-x toggle-debug-on-error` and report an issue with the backtrace.

2. It is also a good idea to turn on Indium's log mode with `M-: (setq indium-client-debug t)`, and attach to the issue report the contents of the `*indium-debug-log*` buffer to help resolve the issue.

Attaching the contents of the `*indium-process*` buffer can help as well in case an error happens in the server process.

1.7.2 The Indium server doesn't start

First, make sure that `indium` is installed as documented in the *The Indium server* section.

If you encounter the error:

```
"error in process filter: Indium server process error: env: node: No such file or␣
↪directory"
```

Chances are that `node` is not in Emacs' `exec-path`. In this case, you can fix it by appending the correct directory to `exec-path`, or use the exec-path-from-shell package:

```
(require 'exec-path-from-shell)
(exec-path-from-shell-initialize)
```

1.7.3 Breakpoints are not set (not using sourcemaps)

If breakpoints do not work, chances are that the project is not configured correctly.

Note: Indium needs to know how to map script source urls to files on disk. It uses the `root` (alias `webRoot`) configuration option as the base path, as described in the *General configuration* page.

Indium provides a command `indium-list-script-sources` to list all scripts parsed by the backend, displaying their sources mapped to files on disk. Check that the file where you're trying to add a breakpoint is listed.

1.7.4 Breakpoints and debugging do not work (using sourcemaps)

Correctly mapping sourcemaps to file locations can be tedious (see *Using sourcemaps*).

Indium provides the command `indium-list-sourcemap-sources` to help configuring sourcemaps correctly. This command displays a list of all sourcemap sources in the runtime as file paths on disk. Check that your files are listed there.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search