Indium Documentation

Release 1.1.0

Nicolas Petton

Jan 18, 2018

Contents

1 Table of contents **3**

1.1 Installation . . . 3
1.2 Getting up and running . . . 4
1.3 The REPL . . . 6
1.4 Interaction in JS buffers . . . 7
1.5 The stepping debugger . . . 9
1.6 The inspector . . . 10
1.7 Network . . . 10
1.8 Listing parsed scripts . . . 11

2 Indices and tables **13**

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.

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 25.1+`, `Chrome 54.0+` (debugging protocol `v1.2`) and `NodeJS 7+`.

Indium is availabe on MELPA, MELPA Stable.

1.1.1 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]
```

1.1.2 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 NodeJS

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

Installing a recent version of NodeJS

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.

Executing NodeJS from Emacs

Indium can start NodeJS processes and connect to them automatically. This is the preferred way of using Indium with NodeJS.

Start a process with `M-x indium-run-node`. Once the process is ready, Indium will connect to it and open a REPL buffer.

The output from the NodeJS process is appended to the `*nodejs process*` buffer.

When a nodejs process has been started with `indium-run-node`, it can be restarted with `indium-restart-node`.

Note: Indium will append the `--inspect` flag to the command-line arguments automatically, so you do not need to provide them.

If you wish to break the execution at the first statement, set `indium-nodejs-inspect-break` to `t`.

Connecting to a Nodejs process

To connect to an existing NodeJS process, make sure that process was started with the `--inspect` flag:

```
node --inspect myfile.js
```

If you wish to break on the first line of the application code, start node using:

```
node --inspect --debug-brk myfile.js
```

Node will tell you to open an URL in Chrome:

```
chrome-devtools://inspector.html?...&ws=127.0.0.1:PORT/PATH
```

Evaluate `M-x indium-connect-to-nodejs RET 127.0.0.1 RET PORT RET PATH`, `PORT` and `PATH` are the ones from the *ws* parameter of the above URL.

Connecting Indium to the node process will open a debugger on the first line of the application code if you passed the CLI argument `--debug-brk`.

1.2.2 Chrome/Chromium

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

Start Chrome/Chromium with the `--remote-debugging-port` flag like the following:

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

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.

To connect to a tab, run from Emacs:

```
M-x indium-connect-to-chrome
```

1.2.3 Using local files when debugging

Indium can use local files when debugging, or to set breakpoints.

Hint: When using `NodeJS`, or when the connected tab uses the `file://` URL, Indium will by itself use local files from disk. In this case there is nothing to setup.

If the Chrome connection uses the `http://` or `https://` protocol, you will have to tell Indium where to find the corresponding JavaScript files on disk by setting up a workspace.

To do that, place an empty `.indium` marker file in the root folder where your **web server serves static files**.

The root folder where the `.indium` file should be put is not always the directory that contains your JavaScript files. It should be the root folder containing static files. Most of the time, it is at least one level above.

Given the following project structure:

```
project/ (current directory)
   www/
      index.html
      css/
         style.css
      js/
         app.js
      .indium
```

Indium will lookup the file `www/js/app.js` for the URL "http://localhost:3000/js/app.js".

Warning: In order for this setup to work, make sure to call `indium-connect-to-chrome` from somewhere within the workspace directory!

1.2.4 Configuring Webpack for the debugger

When Webpack is used to bundle JavaScript files, it is currently required to configure it to emit absolute file paths for sourcemaps, see *Sourcemaps and Webpack*.

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 *Using local files when debugging*).

- `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 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)

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.

1.4.4 Live code update (hot-swapping JavaScript sources)

Indium supports live code updates without the need to reload the browser tab or restart the nodejs process.

This feature works with by hot-swapping the script source, and works even with lexical closures.

Note: This feature currently in only available for Chrome & Chromium.

To enable live updates, make sure Indium is set up to use local files (see *Using local files when debugging*).

- `C-c C-k`: Updates the runtime JavaScript source with the contents of the current buffer (this is also done automatically when a buffer is saved).

You can setup a hook to be run after each script update. For example

```
(add-hook 'indium-update-script-source-hook
          (lambda (url)
            (indium-eval (format "window.dispatchEvent(new CustomEvent('patch',
↪{detail: {url: '%s'}}))"
                                 url))))
```

Then you can use it in your app for development purposes

```
window.addEventListener("patch", (event) => {
  console.log("Patched @ " + new Date().toTimeString().substring(0, 8), event.detail.
↪url);
  // rerender, etc
});
```

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 *Using local files when debugging*).

If you wish to disable sourcemaps when debugging, set `indium-script-enable-sourcemaps` to `nil`.

1.5.2 Sourcemaps and Webpack

When using Webpack to bundle JavaScript files, Indium will only be able to use sourcemaps if Webpack is configured to emit absolute file paths.

Here is an example configuration snippet to be inserted in *webpack.config.json*

```
...
output : {
  ...
  devtoolModuleFilenameTemplate: '[absolute-resource-path]',
  devtoolFallbackModuleFilenameTemplate: '[absolute-resource-path]?[hash]'
}
```

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 Network

You can disable or enable network cache using the following commands:

```
M-x indium-v8-disable-cache
M-x indium-v8-enable-cache
```

Both commands save your choice which will be used for future Indium connections for the current Emacs session.

You can make the cache setting permament by setting *indium-v8-cache-disabled*:

```
(setq indium-v8-cache-disabled t)
```

1.8 Listing parsed scripts

Indium includes the command `indium-list-scripts` to list all JavaScript scripts parsed by the runtime. When using a workspace, local file can be visited from entries in the list (see *Using local files when debugging*).

1.8.1 Using the listing buffer

Here is a list of available keybindings in an script listing buffer:

Keybinding	Description
`RET`	Follow a link open the local file associated with the script
`g`	Update the listing buffer
`n` or `TAB`	Jump to the next script
`p` or `s-TAB`	Jump to the previous script

CHAPTER 2

Indices and tables

- genindex
- modindex
- search