

# honu egg

---

A parser for 'honu' syntax  
Extension for Chicken Scheme  
Version 1.0

felix

---

# Table of Contents

|          |                                         |          |
|----------|-----------------------------------------|----------|
| <b>1</b> | <b>About this egg .....</b>             | <b>1</b> |
| 1.1      | Version history .....                   | 1        |
| 1.2      | Usage .....                             | 1        |
| <b>2</b> | <b>Documentation .....</b>              | <b>2</b> |
| 2.1      | Numbers .....                           | 2        |
| 2.2      | Identifiers .....                       | 2        |
| 2.3      | Strings .....                           | 2        |
| 2.4      | Characters .....                        | 2        |
| 2.5      | Parentheses, Brackets, and Braces ..... | 3        |
| 2.6      | Comments .....                          | 3        |
| 2.7      | Interface .....                         | 3        |
| <b>3</b> | <b>License .....</b>                    | <b>4</b> |
|          | <b>Index .....</b>                      | <b>5</b> |

# 1 About this egg

## 1.1 Version history

1.0            Initial release

## 1.2 Usage

Load this egg like so:

```
(require-extension honu)
```

## 2 Documentation

Honu syntax resembles Java, instead of Scheme. Like Scheme, however, Honu has no fixed syntax. Honu supports extensibility through macros and a base syntax of H-expressions, which are analogous to S-expressions.

Ignoring whitespace, an H-expression is either

- a number
- an identifier
- a string
- a character
- a sequence of H-expressions between parentheses
- a sequence of H-expressions between square brackets
- a sequence of H-expressions between curly braces
- a comment followed by an H-expression
- `#;` followed by two H-expressions
- `#hx` followed by an H-expression
- `#sx` followed by an S-expression

Whitespace for H-expressions is as in Scheme: any character for which `char-whitespace?` returns true counts as a whitespace.

### 2.1 Numbers

The syntax for Honu numbers is the same as for Java. The S-expression encoding of a particular H-expression number is the obvious Scheme number.

### 2.2 Identifiers

The syntax for Honu identifiers is the union of Java identifiers plus semicolon (`;`), comma (`,`), and a set of operator identifiers. An **operator identifier** is any combination of the following characters:

`+ - _ = ? : < > . ! % ^ & * / ~ |`

The S-expression encoding of an H-expression identifier is the obvious Scheme symbol.

Input is parsed to form maximally long identifiers. For example, the input `int->int;` is parsed as four H-expressions: `int`, `->`, `int`, and `;`.

### 2.3 Strings

The syntax for an H-expression string is exactly the same as for an S-expression string, and an H-expression string is represented by the obvious Scheme string.

### 2.4 Characters

The syntax for an H-expression character is the same as for an H-expression string that has a single content character, except that a single quote (`'`) surrounds the character instead of double quotes (`"`). The S-expression representation of an H-expression character is the obvious Scheme character.

## 2.5 Parentheses, Brackets, and Braces

A parenthesized `()`, bracketed `[]`, or braced `{}` H-expression sequence is represented by a Scheme list. The first element of the list is `;%parens` for a parenthesized sequence, `;%brackets` for a brackets sequence, or `;%braces` for a braced sequence. The remaining elements are the Scheme representation for the parenthesized, bracketed, or braced H-expressions in order.

## 2.6 Comments

An H-expression comment starts with either `//` or `/*`. In the former case, the comment runs until a linefeed or return. In the second case, the comment runs until `*/`, but `/* . . . . */` comments can be nested. Comments are treated like whitespace.

`%;` starts an H-expression comment, as in Scheme. It is followed by an H-expression to be treated as white. Note that `%;` is equivalent to `#sx#;#hx`.

## 2.7 Interface

`read-honu` [procedure]  
`(read-honu [PORT [SHOW-LINE-NUMBERS]])`

Reads a single honu expression from `PORT`, which defaults to the value of `(current-input-port)`. If `SHOW-LINE-NUMBERS` is given and true, then parsing errors will report the offending line-number (as obtained by `port-position`).

### 3 License

Copyright (c) 2006, Felix L. Winkelmann. All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the Software), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED ASIS, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Index

read-honu..... 3