

# jni egg

---

An interface to the Java(tm) VM  
Extension for Chicken Scheme  
Version 0.1

Daishi Kato and 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	Limitations .....	3
<b>3</b>	<b>License .....</b>	<b>4</b>
	<b>Index .....</b>	<b>5</b>

# 1 About this egg

## 1.1 Version history

0.1          Initial release

## 1.2 Usage

Load this egg like so:

```
(require-extension jni)
```

## 2 Documentation

This extension uses the Java(tm) native interface (JNI) to run a Java(tm) from Scheme code.

To build this extension you should set some environment variables before invoking `chicken-setup`:

- for GCJ, set `CHICKEN_JNI_USE_GCJ` to some non-empty value
- for Linux, set `JNI_H_PATH` and `JNI_LIB_PATH` to the directories where `jni.h` and `libvjm` are situated

`define-java-classes` [macro]

`(define-java-classes NAME ...)`

Defines the variables `NAME ...`, holding pointers to the Java classes of the same name. `NAME` may be either a symbol or a list of the form `(VARIABLE CLASSNAME)`.

`(define-java-classes Foo Bar)` is equivalent to `(begin (define Foo (jni:find-class "Foo")) (define Bar (jni:find-class "Bar")))`

The following JNI procedures are provided:

```
(jni:find-class STRING) -> CLASS
(jni:define-class STRING LOADER BYTE-VECTOR LENGTH) -> CLASS
(jni:get-version) -> INT
(jni:get-superclass CLASS) -> SUPERCLASS
(jni:is-assignable-from? CLASS SUPERCLASS) -> BOOL
(jni:throw THROWABLE) -> INT
(jni:throw-new CLASS STRING) -> INT
(jni:exception-occurred) -> THROWABLE
(jni:exception-describe)
(jni:exception-clear)
(jni:fatal-error STRING)
(jni:is-same-object? OBJECT1 OBJECT2) -> BOOL
(jni:get-object-class OBJECT) -> CLASS
(jni:is-instance-of? OBJECT CLASS) -> BOOL
(jni:exception-check) -> BOOL
(jni:monitor-enter OBJECT) -> INT
(jni:monitor-exit OBJECT) -> INT
(jni:get-array-length OBJECT) -> INT
(jni:new-object-array LENGTH CLASS OBJECT) -> OBJECT'
(jni:get-object-array-element OBJECT' INT) -> OBJECT
(jni:set-object-array-element! OBJECT' INT OBJECT)
(jni:string->jstring STRING) -> OBJECT
(jni:jstring->string OBJECT) -> STRING
```

Arguments named `OBJECT`, `CLASS` and `THROWABLE` should be Java objects.

`jni:object?` [procedure]

`(jni:object? X)`

Returns `#t` if `X` is a Java object or `#f` otherwise.

`jni:method` [procedure]

(`jni:method` RETURN-TYPE CLASS NAME TYPELIST [SAFE?])

Returns a procedure that, when called, will invoke the Java method `NAME` (which should be a string or symbol) and that has the result-type `RETURN-TYPE` and the argument-types given in `TYPELIST`. `CLASS` should be a Java object representing a class. If `SAFE?` is given and true, then the method may invoke callbacks into Scheme. The returned procedure takes the receiver object as its first argument, followed by zero or more method arguments.

`jni:nonvirtual-method` [procedure]

(`jni:nonvirtual-method` RETURN-TYPE CLASS NAME TYPELIST [SAFE?])

Similar to `jni:method` but returns a procedure that will call a method looked up in a specific class.

`jni:static-method` [procedure]

(`jni:static-method` RETURN-TYPE CLASS NAME TYPELIST [SAFE?])

Similar to `jni:method` but returns a procedure that will call a static method. The procedure does not take an additional argument beyond the arguments that are expected by the static method.

`jni:constructor` [procedure]

(`jni:constructor` CLASS TYPELIST [SAFE?])

Similar to `jni:method` but returns a procedure that will call a constructor method.

`jni:field` [procedure]

(`jni:field` CLASS TYPE NAME)

Returns a procedure that can be used to access the field `NAME` (a string or symbol). The accessor-procedure accepts a single argument: the instance from which the field value should be retrieved. The returned procedure has a setter, so to modify the field, execute (`set! (<ACCESSOR> <OBJECT>) <VALUE>`).

`jni:static-field` [procedure]

(`jni:static-field` CLASS TYPE NAME)

Returns a procedure that can be used to access the static field `NAME` (a string or symbol). The accessor-procedure takes zero arguments. The returned procedure has a setter, so to modify the field, execute (`set! (<ACCESSOR>) <VALUE>`).

## 2.1 Limitations

- There appear to be stack-related problems on certain platforms, which are currently not entirely understood. For this reason the `jni` extension should be considered alpha quality.
- Primitive arrays are not supported.
- No exception handling.

### 3 License

Copyright (c) 2005, Daishi Kato and Felix 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

### D

define-java-classes ..... 2

### J

jni:constructor ..... 3

jni:field ..... 3

jni:method ..... 3

jni:nonvirtual-method ..... 3

jni:object? ..... 2

jni:static-field ..... 3

jni:static-method ..... 3