

# JSE Reference Manual v3

for release 0.10

Jonathan Bachrach  
MIT AI Lab

January 7, 2002

## 1 Introduction

In this manual we describe a macro facility, called the Java Syntactic Extender (JSE) [1], with the superior power and ease of use of Lisp macro systems, but for Java, a language with a more conventional algebraic syntax. We have written a Java implementation which is a preprocessor taking as input files with Java macro definitions and uses and producing pure Java. It uses a parser generator called ANTLR [2] and an associated extended ANTLR Java grammar to produce a Java text to code quote translator. JSE is made available under the GNU Public License (GPL).

## 2 Notation

Throughout this document JSE objects are described with definitions of the following form:

Name	Signature	$\mathcal{N}$
------	-----------	---------------

*Documentation*

where the rightmost kind field has a one letter code as follows:

N	Notation	$\mathcal{N}$
F	Flag	$\mathcal{N}$
S	Static Method	$\mathcal{N}$
C	Constant	$\mathcal{N}$
M	Method	$\mathcal{N}$
W	Work Around	$\mathcal{N}$

## 3 Installation

JSE is currently distributed as a gzipped tar file. Unpack the file as follows:

```
zcat jse-0-9.tar.gz | tar xpf -
```

This will create the following directory hierarchy:

```
jse
  antlr-2.6.0
```

```
antlr -> antlr-2.6.0
DOC
SRC
  EXAMPLES
  TESTS
```

In order to successfully install JSE, you need to make sure your `CLASSPATH` includes directories with ANTLR, JSE, and any macros you might be needing. For `tcsh` for example, I put the following:

```
setenv JSE_ROOT ${HOME}/jse
setenv ANTLR_ROOT ${JSE_ROOT}/antlr
setenv JSE_SRC ${JSE_ROOT}/SRC
setenv CLASSPATH \
  ${ANTLR_ROOT}: \
  ${JSE_SRC}/EXAMPLES: \
  ${JSE_SRC}/TESTS: \
  /usr/local/java: \
  .
```

in my `.tcshrc` file. The `JSE_ROOT` and the Java system directory included in the above `CLASSPATH` will depend on your actual Java installation.

A 2.6.0 version of ANTLR is provided in `ANTLR_ROOT`. JSE will be upgraded to work with the latest version of ANTLR in the near future.

### 3.1 Building JSE

There is a `Makefile` in the `JSE_SRC` directory for rebuilding the system. Type `make` in that directory to build JSE. This will build the JSE system and all of its examples.

## 4 Usage

From the command line you can preprocess a JSE file with the following command:

```
java JSE file.jse ...
```

which will translate `file.jse` to `file.java`. Then you just run your Java compiler on the output as follows:

```
javac file.java ...
```

which will produce `file.class`.

Remember that macros must be compiled before you use them. Also make sure to remove macro class files before recompiling their definitions.

## 4.1 Flags

JSE takes a few flags:

<input type="checkbox"/> -pretty	<input type="checkbox"/> off	<input type="checkbox"/> $\mathcal{F}$
<i>pretty print output</i>		
<input type="checkbox"/> -lineup	<input type="checkbox"/> on	<input type="checkbox"/> $\mathcal{F}$
<i>maintain source line positions</i>		
<input type="checkbox"/> -compile	<input type="checkbox"/> off	<input type="checkbox"/> $\mathcal{F}$
<i>runs javac on resulting java files</i>		
<input type="checkbox"/> -stdio	<input type="checkbox"/> off	<input type="checkbox"/> $\mathcal{F}$
<i>take input from stdin and output to stdout</i>		
<input type="checkbox"/> -deep	<input type="checkbox"/> on	<input type="checkbox"/> $\mathcal{F}$
<i>macro expands recursively</i>		
<input type="checkbox"/> -one	<input type="checkbox"/> off	<input type="checkbox"/> $\mathcal{F}$
<i>macro expands only one level</i>		
<input type="checkbox"/> -trace	<input type="checkbox"/> off	<input type="checkbox"/> $\mathcal{F}$
<i>turn on very verbose tracing</i>		

## 4.2 Example

One simple example is to compile and test the `unless` macro:

```
> java JSE -compile \
    EXAMPLES/unlessSExpander.jse
> java JSE -stdio
unless (true) doit();
eof
if (! true) doit () ;
>
```

In order to recompile the `unless` macro you would do the following:

```
> rm EXAMPLES/unlessSExpander.class
> java JSE -compile \
    EXAMPLES/unlessSExpander.jse
```

I am hoping to make this less cumbersome in the future.

## 4.3 Makefiles

You can write Makefiles to do the JSE preprocessing and Java compilation automatically by including implicit rules for JSE and java suffixed files. For example, in GNU's `Make` you include the following at the top of a Makefile:

```
.SUFFIXES: .java .class .jse
```

to teach it about your new implicit rules and your actual implicit rules at the bottom of your Makefile:

```
.java.class:
$(COMPILER) $(COPTIONS) $?
.jse.java:
java JSE $?
```

## 5 Status

Information and updates will be made available on <http://www.jbot.org/jse>. A paper on JSE can be found in <http://www.jbot.org/jse/jse.pdf>.

JSE is much slower than it can be made to be. Future releases will include efficiency enhancements.

Currently the following are not implemented:

<input type="checkbox"/> tracing	<input type="checkbox"/> -trace	<input type="checkbox"/> $\mathcal{W}$
<input type="checkbox"/> error trailing	<input type="checkbox"/> none	<input type="checkbox"/> $\mathcal{W}$
<input type="checkbox"/> hygiene	<input type="checkbox"/> genSym	<input type="checkbox"/> $\mathcal{W}$
<input type="checkbox"/> default in syntaxSwitch	<input type="checkbox"/> # ...	<input type="checkbox"/> $\mathcal{W}$
<input type="checkbox"/> use of ... outside of codeQuotes	<input type="checkbox"/> * pattern variable	<input type="checkbox"/> $\mathcal{W}$

Please send bug reports or suggestions to [jb@jbot.org](mailto:jb@jbot.org).

## 6 API

The following is a beginning of an API to JSE.

### 6.1 Hierarchy

```
Fragment
  CompoundFragment
  NestedFragment
    BracesFragment
    BracketsFragment
    ParensFragment
  PatternVariableFragment
  DotDotDotFragment
  SequenceFragment
  Expansion
  Template
  LeafFragment
  IdentifierFragment
  LiteralFragment
    CharacterFragment
    StringFragment
    IntegerFragment
    FloatFragment
  PunctuationFragment
  QuestionFragment
  DotFragment
  SeparatorFragment
    CommaFragment
    SemicolonFragment
  SyntaxException
```

SyntaxMatchFailure  
 SyntaxConstraint  
 GrammarSyntaxConstraint  
 List  
 FragmentList  
 LooseFragmentParser  
 MacroExpander  
 PrettyStream

## 6.2 Methods

Here is a partial list of methods. Consult the source code for the rest.

### 6.2.1 Fragment Methods

out	PrettyStream ()	<i>S</i>
<i>pretty printer</i>		
getInt	int ()	<i>M</i>
<i>converts to an integer if possible</i>		
getString	String ()	<i>M</i>
<i>converts to a string if possible</i>		
getFloat	float ()	<i>M</i>
<i>converts to a float if possible</i>		
getValue	Object ()	<i>M</i>
<i>converts to a value if possible</i>		
tokens	List ()	<i>M</i>
<i>returns token representation</i>		
collectBoundVariables	FragmentList ()	<i>M</i>
<i>finds all pattern variables</i>		

### 6.2.2 CompoundFragment Methods

getInsideFragments	FragmentList ()	<i>M</i>
--------------------	-----------------	----------

### 6.2.3 SequenceFragment Methods

FragmentList	SequenceFragment ()	<i>S</i>
--------------	---------------------	----------

### 6.2.4 IdentifierFragment Methods

IdentifierFragment	IdentifierFragment(String)	<i>S</i>
genSym	IdentifierFragment (String name)	<i>S</i>
capitalize	String ()	<i>M</i>
equals	boolean (IdentifierFragment)	<i>M</i>
equals	boolean (String)	<i>M</i>

### 6.2.5 FragmentList Methods

fnil	FragmentList	<i>C</i>
<i>empty list</i>		
flist	FragmentList (Fragment)	<i>S</i>
<i>creates a one fragment list</i>		
felt	Fragment (int)	<i>M</i>
<i>gets nth element</i>		
fpush	FragmentList (Fragment)	<i>M</i>
<i>pushes arg onto front of list</i>		
fpush	FragmentList (int)	<i>M</i>
$\equiv$ fpush(new IntegerFragment(x))		
fpush	FragmentList (float)	<i>M</i>
$\equiv$ fpush(new FloatFragment(x))		
fpush	FragmentList (String)	<i>M</i>
$\equiv$ fpush(new StringFragment(x))		
fpush	FragmentList (char)	<i>M</i>
$\equiv$ fpush(new CharacterFragment(x))		
fhead	Fragment ()	<i>M</i>
<i>gets head of list</i>		
ftail	FragmentList ()	<i>M</i>
<i>gets tail of list</i>		
fappend	FragmentList (FragmentList)	<i>M</i>
<i>concatenation of two lists</i>		
freverse	FragmentList ()	<i>M</i>
<i>reverses list</i>		
tokens	List ()	<i>M</i>
<i>returns token representation</i>		
expandMacros	void (MacroExpander, boolean isRecursive)	<i>M</i>
expandTemplates	void ()	<i>M</i>
pprint	void (PrettyStream)	<i>M</i>
lprint	void (PrettyStream)	<i>M</i>

### 6.2.6 LooseFragmentParser Methods

LooseFragmentParser	LooseFragmentParser (String filename, boolean isTraced)	<i>S</i>
parse	FragmentList ()	<i>M</i>

### 6.2.7 PrettyStream Methods

PrettyStream	PrettyStream (PrintStream)	<i>S</i>
getLine	int ()	<i>M</i>
println	void ()	<i>M</i>
println	void (String)	<i>M</i>
print	void (Object)	<i>M</i>

close	void ()	$\mathcal{M}$
println	void (Object)	$\mathcal{M}$

## References

- [1] Jonathan Bachrach and Keith Playford. The java syntactic extender. In *Proceedings of OOPSLA '01*, October 2001.
- [2] Terence Parr. Antlr. <http://wwwantlr.org/>, 1999.