# Comments

## *Block Comments*

Block comments will have the following structure, with a leading asterisk on each line

```
/*
* Block comment text starts here…
* and more here…
*/
```

## *Header Comment*

All files will contain a file header block comment that contains, at a minimum:

Name of the author
Date of Creation
Functional Description of the Module

## *In-line Comments*

In-line comments that run on multiple lines should line up, for example

```
…
(for (bind ?i 0) (< ?i 42) (++ ?i) ; Iterate over the meaning of existence
   (+ ?count (bobsYourUncle))      ; Count number of uncles named Bob
   (* ?j (judysYourAunte ?count))  ; Not sure what Judy does with this…
)
```

## *Functions and Rules*

All functions and rules will have a block comment that describes the operation and limitations of the function or rule. Special conditions, such as argument value requirements, must be included in this comment.

## *Functions, rules, if, for, while, etc.*

The closing parenthesis in any code block shall have an in-line comment that repeats the reason for the block, but not necessarily the syntax. This comment is required if the block is longer than 10 lines, otherwise it is optional. Example:

```
(while (> ?itemCount  0)
… A fairly long block of code is in here
…
…
…
…
…
…
… and why are we in here in the first place?
… Oh yes, look at the next line!
) ; while (?itemCount > 0)
```

# Formatting

## *Line Length*

The code should be displayable and not result in word-wrapping problems! The exception is if there is an exceptionally long block of static text being bound to a string variable.

# Naming Conventions

## *Functions, rules and variables*

Functions, rules and variables shall start with a lower case letter and can then be a mixture of lower and upper case letters. The use of the underscore character is to be avoided, except in those rare cases where it increases readability. As with Java and C, the variable names ?i, ?j, and ?k should only be used as loop index counters (with some exceptions for physics and engineering models). Names should be meaningful and should indicate the purpose of the function, variable or class. Use of the letters l (lowercase L) and O (uppercase o) should be avoided because they look too much like one and zero and are NEVER allowed by themselves. Exceptions are, of course, when they are used in a word-like name such as `loopCounter` or `pullFromStack`. Examples:

```
(bind ?iVal 0)
(bind ?myInteger 0)
(bind ?yetAnotherVariableName "Hello World")
(bind ?n 1)
(deffunction aFunctionThatTakesVariableNumberOfArguments ($?args)
  (printout t $?args crlf)
)
```

Unlike most languages JESS allows variables to start with a number. In fact, the variable ?21 is perfectly fine (that is a 2 followed by a lower case L). Such a construct would violate the required naming conventions however, so don't do it.

## *Constants*

Constants shall be in upper case letters and defined as either local or global variables, depending on the scope of their use. The underscore character should be used to improve readability. Example:

```
(bind ?SEC_PER_YEAR 3.15e7)
(defglobal ?*SEC_PER_DAY* = 3600)
```

### Magic Numbers

Special values (aka *Magic Numbers*) shall not be used in the code base unless they are defined as constants.

## *Classes (for use with the Java interface API)*

Calling methods in a Java class from within Jess shall always use the `call` operator in order to clearly distinguish Java calls from native Jess operations. Classes shall start with

an upper case letter and can then be a mixture of lower and upper case letters. The use of the underscore character is to be avoided, except in those rare cases where it increases readability. Example:

```
public class Bicycle
{
   … All the stuff you find in a class
} // public class Bicycle
```

### *Indenting and alignment*

Tabs shall not be used for indenting in order to facilitate portability. The standard indent shall be three (3) spaces.

The first parenthesis of a function, rule, conditional, or loop construct shall be adjacent to the name of the construct. The final closing parenthesis shall be on its own line. Example:

```
(deffunction min (?v1 ?v2)
   (if (< ?v1 ?v2) then (bind ?minval ?v1)
    else (bind ?minval ?v2)
   )

   (return ?minval)
)
```

The same convention shall be used for `while`, `for`, and `if` constructs.

### *White Space*

White space is to be used to increase readability. Spaces should be used in conditional and loop statements to improved readability. Blank lines should precede and follow logically associated blocks of code. All functions and rules must have bounding blank lines above and below them.

## Flow Control

The `break` statement shall NOT be used. The `continue` statement shall be used sparingly and only in those cases where it improves readability.

FOR and WHILE Loops: The iterator (the loop counter) in a for-loop shall not be modified within the body of the loop. Additionally, a for-loop shall not be used to simulate a while-construct. The for-loop construct shall always execute a predetermined fixed number of iterations.

All functions shall have a (`return`) statement as their last executable line. In general, there shall be one AND ONLY ONE return statement in a function or method (rules should NOT use the return statement). Note that exceptions can be made if the code is easier to understand if it uses multiple return statements. Exceptions can be made for very small functions, but only if clarity is not sacrificed. The return statement shall be

enclosed in parenthesis, even though the construct works without them in this version of JESS.