

---

# **JBranch Constraint Language**

---

September 27, 1999

Sebastian Fischmeister  
{Fischmeister@computer.org}

## Contents

<b>1</b>	<b>Preface</b>	<b>3</b>
<b>2</b>	<b>Overview</b>	<b>3</b>
2.1	The Language . . . . .	3
2.2	The Interpreter . . . . .	4
2.3	Constraint Manager and Constraint . . . . .	4
2.4	Values and Constraints . . . . .	4
2.5	Debugging . . . . .	4
<b>3</b>	<b>Examples</b>	<b>4</b>
3.1	Without Anything . . . . .	5
3.2	Simple Values . . . . .	6
3.3	Method Invocation . . . . .	7
<b>A</b>	<b>Language BNF</b>	<b>8</b>

## Listings

1	The basic blocks needed for using the Constraint Manager . . . . .	4
2	Checking a dummy constraint. . . . .	5
3	Using variables. . . . .	6
4	Invoking methods. . . . .	7

# 1 Preface

This manual describes how to use the JBRANCH CONSTRAINT LANGUAGE in the Gypsy Environment[1, 2]. Section 2 provides an architectural overview of the system. Finally section 3 gives examples for most of the features. In the appendix there is the BNF of the implemented language (see page 8).

## 2 Overview

The JBRANCH CONSTRAINT LANGUAGE has been developed as part of my master-thesis [3]. As a matter of fact it is not finished yet. So this manual is the only source that is currently available for further information about the language and how to use it.

The language is used in the Supervisor-Worker framework for checking constraints for domains like routing, inter-task, data, etc. Thus it has been developed for this specific use it can be used within Java like a small scripting language.

The Appendix A contains the BNF of this language. The language itself is not very complicated yet it can come in very handy if it is used in the right way.

### 2.1 The Language

The main constructs of this language are “*if*” expressions. Every constraint starts with the keyword *if*. Then there is some sort of clause in between the “*if*” and the “*then*”. After this clause there is either a new constraint, starting with another “*if*” or there is a “*begin*”-“*end*” block which contains the actions of this rule. Whenever the rules are correct, then the action list will be parsed and executed. Actions are divided by semicolons (“;”).

This action statements can be either atoms like “*true*” or “*false*” or they be a registered method. The simplest example rule in this language is the following:

```
if true then begin true end
```

For more complicated examples about the language see section 3. It introduces most of the language’s features in easy-to-understand examples.

## 2.2 The Interpreter

The language has been implemented using the Java Parser Generator[4]. This generator uses the visitor pattern for implementing type checkers or interpreters. The JBRANCH CONSTRAINT LANGUAGE interpreter and type checker also uses this pattern. For more details about the structure see [3].

## 2.3 Constraint Manager and Constraint

The constraint manager is a specific interface, which has to be implemented for customizing the existing one. Customizing the constraint manager also allows implementing own languages or using own constraints. Tightly coupled with the constraint manager interface is the constraint interface. This interface has to be implemented if you want to use custom constraints.

## 2.4 Values and Constraints

The constraint manager which comes with the basic classes has some sort of value table, where it stores all values used in the constraints. So for example if in the language the variable “*hostname*” is used then this variable has to be stored in the constraint managers value table. Please see the examples for further details (section 3).

## 2.5 Debugging

As a matter of fact currently debugging is not supported. The best way to test your statements is to simply dump the rule and the value-table as well and check what is wrong.

## 3 Examples

This section describes elaborate examples for using the existing constraint manager and constraints in Java programs. These examples are only valid for the manager that comes with the Gypsy distribution.

```
1 import at.ac.tuwien.infosys.gypsy.constraints.*;
2 import at.ac.tuwien.infosys.gypsy.constraints.jbranch.*;
3
4 public class BasicBlocks {
5
6     // the main constraintmanager
7     private ConstraintManager cm_;
8 }
```

```

10     // one working constraint
11     private JBranchConstraint c_;
12
13     public BasicBlocks () {
14         cm=new JBranchConstraintManager ();
15     }
16 }

```

Listing 1: The basic blocks needed for using the Constraint Manager

Lines one and two include the main packages needed for using the constraint managers. Line seven declares a variable for the main constraint manager. Line 10 declares a variable for one constraints, so it can be used within methods. In the constructor in line eleven the main constraint manager is initialized.

More or less this is the basic block which has to be present in every program using the constraint language.

### 3.1 Without Anything

This example shows the basic how to define and check constraints.

```

1  import at.ac.tuwien.infosys.gypsy.constraints.*;
2  import at.ac.tuwien.infosys.gypsy.constraints.jbranch.*;
3
4  public class WithoutAnything {
5
6      // the main constraintmanager
7      private ConstraintManager cm_;
8
9      // one working constraint
10     private JBranchConstraint c_;
11
12     public WithoutAnything () {
13         cm_=new JBranchConstraintManager ();
14         c_=new JBranchConstraint ("if true then begin true end", null,0);
15         cm_.addConstraint (c_);
16
17         if (cm_.checkConstraints ()) {
18             System.out.println ("everything all right!");
19         } else {
20             System.out.println ("some constraint is not fulfilled");
21         }
22     }
23
24     public static void main (String a []) {
25         new WithoutAnything ();
26     }
27 }

```

Listing 2: Checking a dummy constraint.

Lines one to thirteen are exactly the same as they have been in the BasicBlocks listing. In line fourteen a constraint is declared:

```
if true then begin true end
```

This constraint is added to the constraint manager in line fifteen. In line seventeen a checking operation is performed. If all constraints are fulfilled then “*true*” is returned else “*false*”.

## 3.2 Simple Values

The next example shows the use of user defined values. This time the value-table for the constraint will not be “*null*” anymore, as it was in the last example (see line fourteen).

```
1  import at.ac.tuwien.infosys.gypsy.constraints.*;
2  import at.ac.tuwien.infosys.gypsy.constraints.jbranch.*;
3  import java.util.*;
4
5  public class SimpleValue {
6
7      // the main constraintmanager
8      private ConstraintManager cm_;
9
10     // one working constraint
11     private JBranchConstraint c_;
12
13     public SimpleValue() {
14         cm_=new JBranchConstraintManager();
15
16         Hashtable ht = new Hashtable();
17         ht.put("dada",new Integer(1));
18         ht.put("foo","bar");
19
20         c_=new JBranchConstraint("if (dada==1)&&(foo==\"bar\")"+
21                                 "then begin true end", ht,0);
22         cm_.addConstraint(c_);
23
24         if(cm_.checkConstraints()) {
25             System.out.println("everything is right!");
26         } else {
27             System.out.println("some constraint is not fulfilled");
28         }
29     }
30
31     public static void main(String a[]) {
32         new SimpleValue();
33     }
34 }
```

Listing 3: Using variables.

In line sixteen and seventeen the hashtable for the constraint is initialized and the variable value is inserted. In line twenty there is the constraint using both variables.

If the variables should be updated for the constraint use the *updateValues()* method of the ConstraintManager interface. The current version does not allow different value tables for each constraint. The only way to do this is the get each

constraint from the manager and set update the values automatically. Furthermore the new value table has to contain all values, because it simply overrides the old one.

### 3.3 Method Invocation

The next example shows how to invoke a method from within a constraint. A method will be registered and invoked during checking the constraint. The method can also be invoked as an action.

```
import at.ac.tuwien.infosys.gypsy.constraints.*;
2 import at.ac.tuwien.infosys.gypsy.constraints.jbranch.*;
import at.ac.tuwien.infosys.gypsy.constraints.jbranch.types.*;
4
import java.util.*;
6
public class SimpleMethod {
8
    // the main constraintmanager
10    private ConstraintManager cm_;

12    // one working constraint
    private JBranchConstraint c_;
14

    public SimpleMethod () {
16        try {
            cm_=new JBranchConstraintManager ();
18
            Hashtable ht = new Hashtable ();
20            ht.put("dada",new Integer(1));

22            ht.put("themethod",
                new MethodCall (this,
24                                this.getClass().
                                    getMethod ("callBackMethod", null)));
26
            c_=new JBranchConstraint ("if themethod() == 11" +
28                                    " then begin true end", ht, 0);
            cm_. addConstraint (c_);
30
            if(cm_. checkConstraints ()) {
32                System.out.println ("everything all right!");
            } else {
34                System.out.println ("some constraint is not fulfilled");
            }
36        } catch (Exception e) {
            e.printStackTrace ();
38        }

40    }

42    public Integer callBackMethod () {
        System.out.println ("wanna test this method!");
44        return new Integer(11);
    }
46

    public static void main(String a[]) {
```

```
48     new SimpleMethod ();
      }
50 }
```

Listing 4: Invoking methods.

In the lines 22 to 25 the method call is initialized and stored in the value table. Using the method invocation in a constraint is nearly the same as using a value. The difference is that a pair of braces “()” has to be put afterwards, so the interpreter knows about it. In general any method can be registered and also values can be used for the method call. However, this version does not support using values from within the constraint.

One more word about side effects and short circuits: First of all when a method is executed it is the same as if the method will be executed from within normal Java code. So it has the same restrictions and also the same abilities. It **can** modify other objects! And second, short circuits do not exist. So when for example the first part of an expression already evaluated to false then the second part will still be executed.

## A Language BNF

### Syntax Definition

The syntax of the BNF given below follows these rules:

[ X ]	X may occur
( X ) ?	X may occur
( X ) *	X occurs without limits
( X ) +	X occurs at least once

### Language Definition

The constraint parser skips the following symbols: “\n”, newline, carriage return and tab.

**Reserved words** are:

"if", "else", "then", "begin", "end", "true", "false"

**Operators** are:

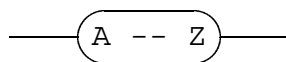
EQ : "=="  
LT : "<"  
LEQ: "<="  
GT : ">"  
OR : "||"  
GEQ: ">="  
AND: "&&"  
NEQ: "!="  
NOT: "!"  
DIV: "/"

**Single Tokens** are:

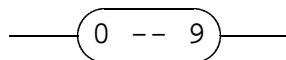
COLON : ":"  
POINT : "."  
MINUS : "-"  
PLUS : "+"  
SEMICOLON : ";"  
QUOTES : "\""  
SBRACKET : "("  
CBRACKET : ")"  
UNDERSCR : "\_"

**Tokens** are:

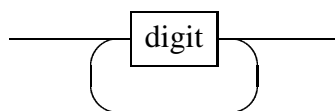
*letter*



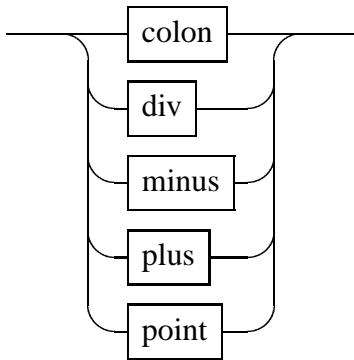
*digit*



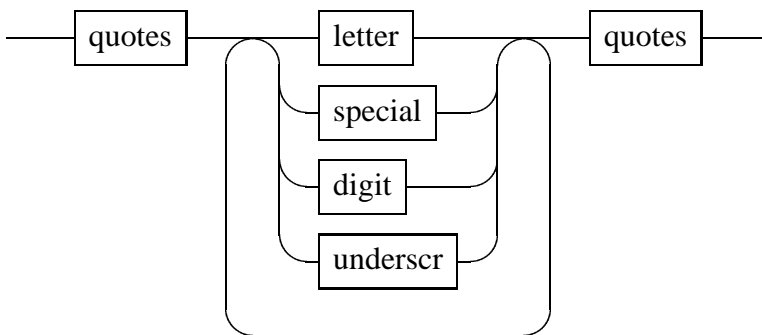
*number*



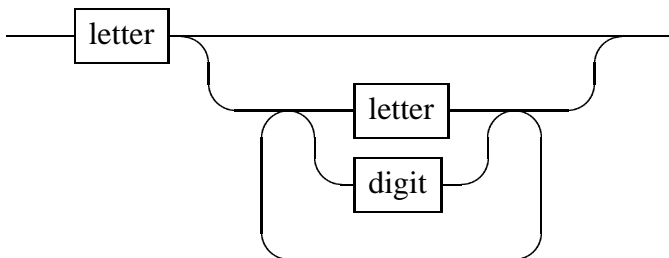
*special*



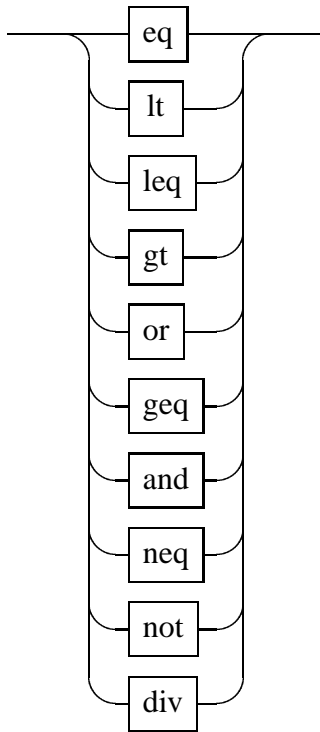
*string*



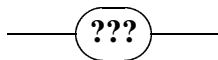
*variable*



*operator*



The EBNF for the language is now:



## References

- [1] M. Jazayeri and W. Lugmary, "Gypsy: A Component-based Approach to Mobile Agent Systems," Tech. Rep. TUV-1841-99-09, Distributed Systems Group, Technical University of Vienna, Argentinierstrasse 8, A-1040 Vienna, Austria, 1999.
- [2] W. Lugmary, "The Gypsy Environment." World Wide Web Site, 1998. <http://www.infosys.tuwien.ac.at/Gypsy/>.
- [3] S. Fischmeister, "An Effective Architecture for Secure Mobile Agent Design: The Supervisor – Worker Framework," Master's thesis, Technical University of Vienna, Vienna, Austria, 1999.

[4] "Java Parser Generator." World Wide Web Site. <http://www.metamata.com/JavaCC/>.