The HBA Language

A Simple Text-to-HTML converter for Host Bus Adapter Reports

YI GONG

December 19, 2005

Table of contents

1. Introduction

The HBA language is designed to help users to convert the text records into HTML table format. The syntax of HBA language is simple. With several lines of HBA codes, the users are able to create a report in HTML format.

The HBA language is easy to use. With HBA language, the users don't have to know HTML language to generate reports in HTML table format.

The language is named after its typical usage – to generate HTML report for Fibre Channel HBA – Host Bus adapter. To implement an enterprise SAN (Storage Area Network) solution, an important step is to identify the driver and firmware versions of FC HBA, and make sure they are on the support matrix.

Storage vendors such as EMC and IBM usually publish support matrix, on which lists all the supported driver and firmware versions. The listed driver/firmware versions have been thoroughly tested and therefore fully supported by the storage vendors. To avoid un-necessary troubleshooting, the first step for SAN implementation is usually to make sure the driver and firmware versions are on the verdor's support matrix. The HBA language will help field engineer to create HTML report of HBA very easily.

1.1 Features

Text Parsing

HBA allows a user to search through a set of data using the functions provided. The user will be able to state the record name in the command input file.

HTML Handling

HBA has the ability to describe HTML output clearly, and handle the HTML tags.

Data Comparison

HBA allows a user to compare the value of text string in the text file with the pre-defined string. If the actual string is different form the pre-defined string, the user can specify the actions, usually to append extra HTML table cells to HBA report.

1.2 HBA Sample Syntax

Text parsing

```
//sample HBA syntax
START;                  //start of the program
```

INFILE = “….” ; //text file name
OUTFILE =”…” ; //output html file name
DELIMITER = “...”; //the delimiter to separate the text records

Data Comparison

//HBA code
IF NAME = "…" THEN VALUE = "…"; //expected value for specific record

HBA language will search for the record in the text file and compare the value with expected value. Related HTML output will be generated.

2. Tutorial

2.1 An example

A sample program file “sample.hba” is provided for tutorial purpose.

```
//sample.hba
//start of the program

START;

// input file in text format

INFILE = "text.txt";

// output file in HTML format, which is converted from input text file

outfile = "output.html";

// delimiter for the records in the text input file

delimiter = ":";

// the name of the record which HBA language should search in input
// text file; and the expected value. In this example, HBA will look for
// a record which contains string "Firmware" and compare the related firmware
// version with the "3.03.06". Keywods could be lower-case as well.

IF NAME = "Firmware" THEN VALUE = "3.03.07";
```

The sample program converts “text.txt” into “output.html”, which is in HTML table format.

//text.txt

```
Host                  : DC2MS906
Adapter Number        : 0
Adapter Model         : QLA2340
Adapter Node Name     : 20-00-00-E0-8B-12-66-CF
Adapter Port Name     : 21-00-00-E0-8B-12-66-CF
Adapter Port ID       : 00-00-00
Serial Number         : M32742
Driver Version        : SCSIport 9.0.0.2 (w32 IP)
BIOS                  : 1.42
Firmware Version      : 3.03.06
Device Target Count   : 0
PCI Bus Number        : 1
PCI Slot Number       : 2
PortType (Topology)   : NPort
Adapter Status        : Loop down
```

The generated HTML table will be :

Host	DC2MS906
Adapter Number	0
Adapter Model	QLA2340
Adapter Node Name	20-00-00-E0-8B-12-66-CF
Adapter Port Name	21-00-00-E0-8B-12-66-CF
Adapter Port ID	00-00-00
Serial Number	M32742
Driver Version	SCSIport 9.0.0.2 (w32 IP)
BIOS	1.42
Firmware Version	3.03.06
Device Target Count	0
PCI Bus Number	1
PCI Slot Number	2
PortType (Topology)	NPort
Adapter Status	Loop down

The firmware version should be 3.03.07

The following are the steps how to run “sample.hba” in a HBA environment:

- open a DOS prompt, go to the directory where smple.hba resides
- run “java HBA sample.hba”
- you will find the output.html file under the current directory

2.2 Installation steps

HBA language is developed for Windows environment. You need the followings software to run HBA language properly.

Windows 2000/XP
JAVA J2SE v1.4.2 or higher
HBA language package

HBA language installation steps:

- unzip the hba.zip to C:\HBA directory
- on DOS prompt, run “set classpath=%classpath%;C:\HBA”

3. Language Reference Manual

3.1 Lexical conventions

3.1.1 Comments

A Java style comment is supported. The characters // introduce a comment, which terminates with the new line. // has no special meaning inside comment line.

3.1.2 Whitespace

Whitespace is defined as the ASCII space, horizontal tab.

3.1.3. Identifiers

An identifier is a sequence of letters and digits; the first character must be alphabetic. Upper and lower case letters are considered different. Identifier -> letter ***(*** letter | digit **|** **‘_’*)****.

3.1.4. Keywords

The following identifiers are reserved for use as keywords, and may not be used otherwise:

INFILE
OUTFILE
START
IF
THEN
NAME
VALUE
DELIMITER

3.1.5 Numbers

A number consists of digits.

3.1.6 Strings

A string is a sequence of characters enclosed by double quotes ‘ " ‘. A double quote inside the string is represented by two consecutive double quotes.

3.1.7 Separators

The following ASCII characters are separators:

(
)
;

3.1.8. Built-in variables

The following identifiers are reserved for use as built-in variables, and may not be used otherwise:

Fin	Command input file name
Fout	HTML format file name generated by HBA
DeLi	Delimiter specified by users
Fnm	Specific field name
Fvlu	Expected field value

3.2 Expressions

Primary expressions include identifiers and constants.

3.2.1 Identifier

An identifier is a primary expression.
.

3.2.2 Constant

A constant is a right-value expression, which will be evaluated to the constant itself.

3.3 Operators

3.3.1 Assignment operators

variable = expression

The = (assign to) operator groups right-to-left. The value of the expression replaces the object referred to by the variable. The operands need to have the string constant type.

3.4 Statements

Statements are executed in sequence.

3.4.1 Expression statement

Statements are in the form of expression.

3.4.2 Conditional statement

if (expression) then statement

The expression is evaluated first, and if it is non-zero, the statement is executed, which is a expression as well.

4. Project plan

4.1 Software development environment

Windows XP/2000
Java SDK 1.4.2
ANTLR 2.7.5

4.2 Project logs

October 8, 2005 Identify the functions of this new language

October 10, 2005 Break down the functions of front-end and backend modules

October 20, 2005 Set up software development environment

October 23, 2005 First version of white paper

October 25, 2005 Front-end started

November 5, 2005 Back-end started

November 19, 2005 TreeWalker tested

November 26, 2005 First version of front-end and back-end

December 3, 2005 Modifications on back-end module

December 12, 2005 Tests started

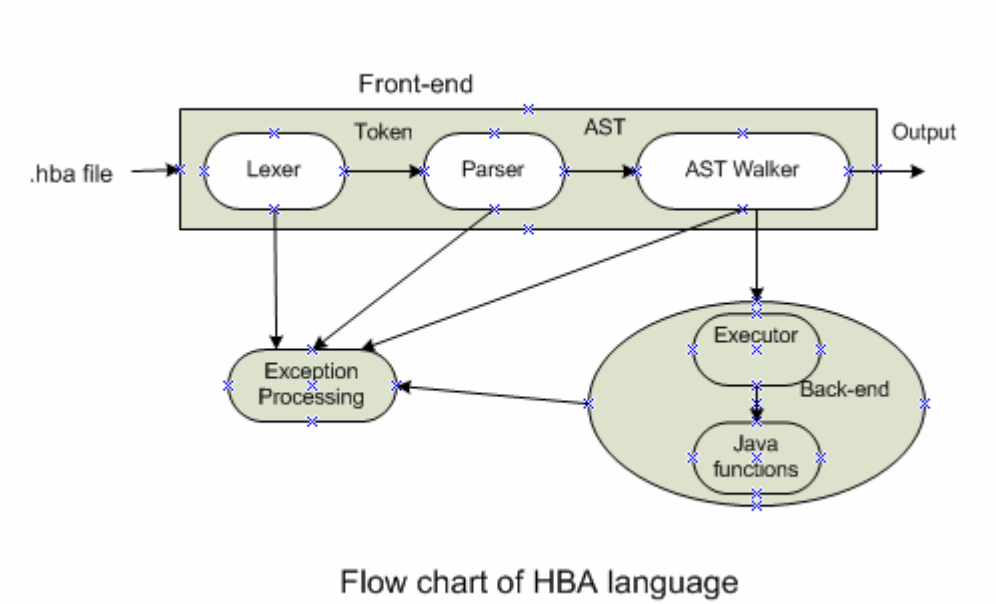
December 14, 2005 Major tests done

December 15, 2005 Started final report

December 19, 2005 Final release

5. Architecture Design

The HBA language interpreter consists of three components, a lexer which reads the program files to tokens, a parser that analyzes the syntactic structure of the program and converts it to an abstract syntax tree, a tree walker which travels the abstract syntax tree and calls corresponding functions, a module that looks up symbol tables and diverts operations to a library written in Java.



Flow chart of HBA language

The lexer, parser and tree walker are implemented by ANTLR. The lexer, parser and tree walker are implemented in Antlr source file hba.g,. The tree walker calls the HBAfunc method in class HBALib. The main() method in the class HBA initializes the lexer, the parser and the tree walker. The class HBA also handles exceptions and errors, and prints corresponding messages.

The Front-End process starts with the input of a HBA command source file. The command file has .hba extension, functions written in Java checks the extension of this file in the main() method of class HBA.

The Back-End starts with the AST passed by the parser. It calls the HBAfunc method from the tree walker. The basic function is to convert the text record into HTML table format.

6. Test plan

All the testing work is done by using print statement. The following tests have been done:

1) Make sure the treewalker picks up the right value handed by the parser, and pass them to the java functions

2) This language allows the keywords to be both lower-case and upper-case in the command input file. For example, “if .. then…” can also be “IF … THEN…”. It’s tested in the command input file.

3) Different field names were tested in the command input file, such as “Firmware”, “Driver” or “BIOS”, so the language can cover the major attributes of Host-Bus-Adapter.

4) Java functions were tested to accept the input file and generate HTML format output files.

7. Lesson learned

I worked on my own in this compiler project. The advantage is that there is no misunderstanding between team members. And there is no waiting time, I can work on the project whenever I got time. The drawback I realized is that since there is no input and assistance from the co-workers, if you make mistakes at the design stage, it would cost you days or even weeks to straight it out later. When I started the code writing, I thought that lexer/parser can handle the text-to-html conversion. It did generate html file from text file, but it's very difficult to implement the advanced functions with just lexer/parser, for example to find the specific records and compare the related value with the expected value. So I had to rewrite the text-to-html conversion function with Java. It also proved that design is very important.

Appendix A

Code listing

sample.hba

```
//sample.hba
// this is the command input file

//start of the program

START;

// input file in text format

INFILE = "text.txt";

// output file in HTML format, which is converted from input text file

outfile = "output.html";

// delimiter for the records in the text input file

delimiter = ":";
```

hba.g

```
// HBA language grammar file, developed by ANTLR-2.7.5
```

```
class SimpWalker extends TreeParser;
options{
        importVocab = HBAvoc;
      }

{
  public static String Fin;
  public static String Fout;
  public static String DeLi;
  public static String Fnm;
  public static String Fvlu;


 public void initialize()
 {

   Fin=null;
   Fout=null;
   DeLi=null;
   Fnm=null;
   Fvlu=null;

 }
}

hba_def returns [String s]
{

  s = "a1";

  String t1,t2,t3,f1,f2=null;

}

: #(st:Sstart t1=hba_def t2=hba_def t3=hba_def f1=hba_def )
  {

        HBALib.HBAfunc (Fin, Fout, DeLi, Fnm, Fvlu);

  }



| #(m:INFILE n:STRING)
```

```
        { Fin = n.getText();
//        System.out.println( Fin ); System.out.println();
        }
| #(o:OUTFILE q:STRING)
        { Fout = q.getText();
//        System.out.println( Fout ); System.out.println();
        }
| #(t:DELIMITER r:STRING)
        { DeLi = r.getText();
//        System.out.println( DeLi ); System.out.println();
        }
| #(fw:FW g:STRING)
        { Fvlu = g.getText();
//        System.out.println( Fvlu ); System.out.println();
        }

| #(f:IF h:STRING f2=hba_def )
        { Fnm = h.getText();
//        System.out.println( Fnm ); System.out.println();
        } ;


class SimpParser extends Parser;
options { buildAST = true; k = 2; }

hba_def
        : Sstart^SEMI!
          ifile_stmt
```

```
            ofile_stmt
            de_stmt
            if_then_stmt
        ;

ifile_stmt
        : INFILE^ ASSIGN! STRING SEMI!
        ;

ofile_stmt
        : OUTFILE^ ASSIGN! STRING SEMI!
        ;

de_stmt
        : DELIMITER^ ASSIGN! STRING SEMI!
        ;

then_stmt
        : FW^ ASSIGN! STRING
        ;

if_then_stmt
        : IF^ FIELD! ASSIGN! STRING THEN! then_stmt SEMI!
        ;



class SimpLexer extends Lexer;
options { charVocabulary = '\3'..'\377';
testLiterals = false;
exportVocab = HBAvoc;
k = 2; }


ASSIGN : '=' ;

LPAREN : '(';

RPAREN : ')';

SEMI : ';' ;


protected LETTER : ('a'..'z' | 'A'..'Z') ;

protected DIGIT : '0'..'9' ;
```

```
//ID options { testLiterals = true; }
// : LETTER (LETTER | DIGIT | '_' | '.')* ;

NUMBER : (DIGIT)+;

STRING : ""! ("" ""! | ~(""))* ""! ;

//WS : ( ' ' | '\t' | '\n' { newline(); } | '\r' )
//     { $setType(Token.SKIP); } ;

//COMMENT
//  : ("//") (' '..'~' | '\t')* ('\r') {
//          $setType(Token.SKIP);};

NL
 : ("\r\n" | '\n' | "\r\t") {
   newline();
   $setType(Token.SKIP);
   };

COMMENT
 : ("//") (' '..'~' | '\t')* NL {
   $setType(Token.SKIP);
   };

WS
 : (' ' | '\t') {
   $setType(Token.SKIP);
   };

INFILE : ("INFILE"|"infile");

OUTFILE : ("OUTFILE"|"outfile");

Sstart: ("START"|"start");

IF: ("IF"|"if");

THEN: ("THEN"|"then");

FIELD: ("NAME"|"name");

FW: ("VALUE"|"value");

DELIMITER: ("DELIMITER"|"delimiter");
```

hba.java

```
// main program to initiate lexer, parser and treewalker in HBA language

import antlr.CommonAST;
import java.util.*;
import java.io.*;

class HBA

{
        public static void main(String[] args) {

        extfunc(args[0]);

      FileInputStream fileInput = null;

        try {

         fileInput = new FileInputStream(args[0]);

        } catch(Exception e) { nocmd(); }


    try {
         DataInputStream input = new DataInputStream(fileInput);

      SimpLexer csvLexer =   new SimpLexer(input);
      SimpParser csvParser = new SimpParser(csvLexer);

         csvParser.hba_def();


         CommonAST tree = (CommonAST)csvParser.getAST();

         SimpWalker walker = new SimpWalker();

         walker.initialize();

         String r = walker.hba_def( tree );


      } catch(Exception e) { System.err.println(e.getMessage());}
```

```
}

 private static void extfunc(String cmdf)

{

        boolean exb = false;

        try{

                String ext = "";

                for(int j = 0; j < cmdf.length(); j++)
                  {
                        if(cmdf.charAt(j) == '.')

                        exb = true;

                        if( !exb )

                                continue;

                        else

                                ext = ext + cmdf.charAt(j);

                   }

                if(! ext.equals(".hba"))

                {

                System.out.println("HBA exception: Command File in incorrect format.
HBA only accepts .hba files\n");

                System.exit(0);

                }
        } catch(Exception e) { System.err.println(e.getMessage());}


} //extfunc


 private static void error() {
```

```
    System.out.println("****************************************");
    System.out.println("*   Usuage: java HBA cmdfile           *");
    System.out.println("****************************************");
    System.exit(0);
  }

  private static void nocmd() {
    System.out.println("****************************************");
    System.out.println("* You must give a valid HBA cmdfile  *");
    System.out.println("****************************************");
    System.exit(0);
  }

}
```

HBALib.java

```
// text-to-html conversion functions

import java.io.*;

public class HBALib {

 public static String InFile;

 public static String OutFile;

 public static String FieldName;

 public static String FieldValue;

 public static String ReadLine;

 public static BufferedReader InFileRe;

 public static BufferedWriter OutFileWr;

 public static String p;

 public static String[] result;

 public static String look;

 public static String flag = "green";
```

```
public static void HBAfunc(String InFile, String OutFile, String Delimiter, String
FieldName, String FieldValue){


        htmlfunc(InFile, OutFile, Delimiter, FieldName, FieldValue);

}

public static void htmlfunc(String InFile, String OutFile, String Delimiter, String
FieldName, String FieldValue){
  try{

      File inputFile = new File(InFile);

         File outputFile = new File(OutFile);

      FileReader in = new FileReader(inputFile);

      FileWriter out = new FileWriter(outputFile);

         BufferedReader InFileRe = new BufferedReader(in);

          p ="";
       p+="<html>";
       p+="  <body>";
          p+="  <table align=\"center\" BGCOLOR='wheat' border=\"1\">\n";


        while ((ReadLine = InFileRe.readLine()) != null)
       {
                ReadLine = ReadLine;

                result = ReadLine.split( Delimiter );

                p+="    <tr>\n";

                for (int i =0; i < result.length; i++)

                {

                p+="      <td>";
                      p+=result[i];
                         p+="      <td>\n";

                }
```

```
                if ( result[0].startsWith( FieldName ) ) {

                        look = result[1].trim();

                        if ( look.equals( FieldValue ) ) {

                          look = "This "+FieldName+" version "+look+" is supported";
//                        System.out.println( look );

                        }else{

                                look = "The "+FieldName+" version should be
"+FieldValue;
                                flag = "red";
//                              System.out.println( look );

                           }

                }

        p+="    <tr>\n";

          }
        p+="\n";


        if (flag.equals( "green" )) {

                p+="  <table align=\"center\" BGCOLOR=green border=\"1\">\n";

        }else{

        p+="  <table align=\"center\" BGCOLOR=red border=\"1\">\n";}

        p+="    <tr>\n";

        p+="      <td>";
        p+=look;
        p+="      <td>\n";

        p+="    <tr>\n";

        p+="  </body>";

        p+="</html>";
```

```
        out.write(p);

            in.close();

        out.close();

    }  catch(Exception e) { System.err.println(e.getMessage());}


  }//main


}
```

SimpLexter.java

```
// $ANTLR 2.7.5 (20050201): "hba.g" -> "SimpLexer.java"$

import java.io.InputStream;
import antlr.TokenStreamException;
import antlr.TokenStreamIOException;
import antlr.TokenStreamRecognitionException;
import antlr.CharStreamException;
import antlr.CharStreamIOException;
import antlr.ANTLRException;
import java.io.Reader;
import java.util.Hashtable;
import antlr.CharScanner;
import antlr.InputBuffer;
import antlr.ByteBuffer;
import antlr.CharBuffer;
import antlr.Token;
import antlr.CommonToken;
import antlr.RecognitionException;
import antlr.NoViableAltForCharException;
import antlr.MismatchedCharException;
import antlr.TokenStream;
import antlr.ANTLRHashString;
import antlr.LexerSharedInputState;
import antlr.collections.impl.BitSet;
import antlr.SemanticException;

public class SimpLexer extends antlr.CharScanner implements HBAvocTokenTypes,
TokenStream
 {
```

```
public SimpLexer(InputStream in) {
	this(new ByteBuffer(in));
}
public SimpLexer(Reader in) {
	this(new CharBuffer(in));
}
public SimpLexer(InputBuffer ib) {
	this(new LexerSharedInputState(ib));
}
public SimpLexer(LexerSharedInputState state) {
	super(state);
	caseSensitiveLiterals = true;
	setCaseSensitive(true);
	literals = new Hashtable();
}

public Token nextToken() throws TokenStreamException {
	Token theRetToken=null;
tryAgain:
	for (;;) {
		Token _token = null;
		int _ttype = Token.INVALID_TYPE;
		resetText();
		try {   // for char stream error handling
			try {   // for lexical error handling
				switch ( LA(1)) {
				case '=':
				{
					mASSIGN(true);
					theRetToken=_returnToken;
					break;
				}
				case '(':
				{
					mLPAREN(true);
					theRetToken=_returnToken;
					break;
				}
				case ')':
				{
					mRPAREN(true);
					theRetToken=_returnToken;
					break;
				}
				case ';':
				{
```

```
            mSEMI(true);
            theRetToken=_returnToken;
            break;
}
case '0':  case '1':  case '2':  case '3':
case '4':  case '5':  case '6':  case '7':
case '8':  case '9':
{
            mNUMBER(true);
            theRetToken=_returnToken;
            break;
}
case '"':
{
            mSTRING(true);
            theRetToken=_returnToken;
            break;
}
case '\n':  case '\r':
{
            mNL(true);
            theRetToken=_returnToken;
            break;
}
case '/':
{
            mCOMMENT(true);
            theRetToken=_returnToken;
            break;
}
case '\t':  case ' ':
{
            mWS(true);
            theRetToken=_returnToken;
            break;
}
case 'O':  case 'o':
{
            mOUTFILE(true);
            theRetToken=_returnToken;
            break;
}
case 'S':  case 's':
{
            mSstart(true);
            theRetToken=_returnToken;
```

```
                                        break;
                                }
                                case 'T':  case 't':
                                {
                                        mTHEN(true);
                                        theRetToken=_returnToken;
                                        break;
                                }
                                case 'N':  case 'n':
                                {
                                        mFIELD(true);
                                        theRetToken=_returnToken;
                                        break;
                                }
                                case 'V':  case 'v':
                                {
                                        mFW(true);
                                        theRetToken=_returnToken;
                                        break;
                                }
                                case 'D':  case 'd':
                                {
                                        mDELIMITER(true);
                                        theRetToken=_returnToken;
                                        break;
                                }
                                default:
                                        if ((LA(1)=='I'||LA(1)=='i') &&
(LA(2)=='N'||LA(2)=='n')) {
                                                mINFILE(true);
                                                theRetToken=_returnToken;
                                        }
                                        else if ((LA(1)=='I'||LA(1)=='i') &&
(LA(2)=='F'||LA(2)=='f')) {
                                                mIF(true);
                                                theRetToken=_returnToken;
                                        }
                                else {
                                        if (LA(1)==EOF_CHAR) {uponEOF();
_returnToken = makeToken(Token.EOF_TYPE);}
                                else {throw new
NoViableAltForCharException((char)LA(1), getFilename(), getLine(), getColumn());}
                                }
                                }
                                if ( _returnToken==null ) continue tryAgain; // found SKIP
token
```

```
                        _ttype = _returnToken.getType();
                        _returnToken.setType(_ttype);
                        return _returnToken;
                }
                catch (RecognitionException e) {
                        throw new TokenStreamRecognitionException(e);
                }
        }
        catch (CharStreamException cse) {
                if ( cse instanceof CharStreamIOException ) {
                        throw new
TokenStreamIOException(((CharStreamIOException)cse).io);
                }
                else {
                        throw new TokenStreamException(cse.getMessage());
                }
        }
    }
}

    public final void mASSIGN(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
        int _ttype; Token _token=null; int _begin=text.length();
        _ttype = ASSIGN;
        int _saveIndex;

        match('=');
        if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                _token = makeToken(_ttype);
                _token.setText(new String(text.getBuffer(), _begin, text.length()-
_begin));
        }
        _returnToken = _token;
    }

    public final void mLPAREN(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
        int _ttype; Token _token=null; int _begin=text.length();
        _ttype = LPAREN;
        int _saveIndex;

        match('(');
        if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                _token = makeToken(_ttype);
                _token.setText(new String(text.getBuffer(), _begin, text.length()-
_begin));
```

```
		}
		_returnToken = _token;
	}

	public final void mRPAREN(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = RPAREN;
		int _saveIndex;

		match(')');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-
_begin));
		}
		_returnToken = _token;
	}

	public final void mSEMI(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = SEMI;
		int _saveIndex;

		match(';');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-
_begin));
		}
		_returnToken = _token;
	}

	protected final void mLETTER(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = LETTER;
		int _saveIndex;

		{
		switch ( LA(1)) {
		case 'a':  case 'b':  case 'c':  case 'd':
		case 'e':  case 'f':  case 'g':  case 'h':
		case 'i':  case 'j':  case 'k':  case 'l':
		case 'm':  case 'n':  case 'o':  case 'p':
```

```
		case 'q':  case 'r':  case 's':  case 't':
		case 'u':  case 'v':  case 'w':  case 'x':
		case 'y':  case 'z':
		{
			matchRange('a','z');
			break;
		}
		case 'A':  case 'B':  case 'C':  case 'D':
		case 'E':  case 'F':  case 'G':  case 'H':
		case 'I':  case 'J':  case 'K':  case 'L':
		case 'M':  case 'N':  case 'O':  case 'P':
		case 'Q':  case 'R':  case 'S':  case 'T':
		case 'U':  case 'V':  case 'W':  case 'X':
		case 'Y':  case 'Z':
		{
			matchRange('A','Z');
			break;
		}
		default:
		{
			throw new NoViableAltForCharException((char)LA(1),
getFilename(), getLine(), getColumn());
		}
		}
		}
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-
_begin));
		}
		_returnToken = _token;
	}

	protected final void mDIGIT(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = DIGIT;
		int _saveIndex;

		matchRange('0','9');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-
_begin));
		}
		_returnToken = _token;
```

```
	}

	public final void mNUMBER(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = NUMBER;
		int _saveIndex;

		{
		int _cnt23=0;
		_loop23:
		do {
			if (((LA(1) >= '0' && LA(1) <= '9'))) {
				mDIGIT(false);
			}
			else {
				if ( _cnt23>=1 ) { break _loop23; } else {throw new
NoViableAltForCharException((char)LA(1), getFilename(), getLine(), getColumn());}
			}

			_cnt23++;
		} while (true);
		}
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-
_begin));
		}
		_returnToken = _token;
	}

	public final void mSTRING(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = STRING;
		int _saveIndex;

		_saveIndex=text.length();
		match("");
		text.setLength(_saveIndex);
		{
		_loop27:
		do {
			if ((LA(1)=="") && (LA(2)="")) {
				match("");
				_saveIndex=text.length();
```

```
                        match("");
                        text.setLength(_saveIndex);
                    }
                    else if ((_tokenSet_0.member(LA(1)))) {
                        {
                        match(_tokenSet_0);
                        }
                    }
                    else {
                        break _loop27;
                    }

                } while (true);
                }
                _saveIndex=text.length();
                match("");
                text.setLength(_saveIndex);
                if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                    _token = makeToken(_ttype);
                    _token.setText(new String(text.getBuffer(), _begin, text.length()-
_begin));
                }
                _returnToken = _token;
        }

        public final void mNL(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
                int _ttype; Token _token=null; int _begin=text.length();
                _ttype = NL;
                int _saveIndex;

                {
                if ((LA(1)=='\r') && (LA(2)=='\n')) {
                    match("\r\n");
                }
                else if ((LA(1)=='\r') && (LA(2)=='\t')) {
                    match("\r\t");
                }
                else if ((LA(1)=='\n')) {
                    match('\n');
                }
                else {
                    throw new NoViableAltForCharException((char)LA(1),
getFilename(), getLine(), getColumn());
                }
```

```
                }

                newline();
                _ttype = Token.SKIP;

                if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                        _token = makeToken(_ttype);
                        _token.setText(new String(text.getBuffer(), _begin, text.length()-
_begin));
                }
                _returnToken = _token;
        }

        public final void mCOMMENT(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
                int _ttype; Token _token=null; int _begin=text.length();
                _ttype = COMMENT;
                int _saveIndex;

                {
                match("//");
                }
                {
                _loop33:
                do {
                        switch ( LA(1)) {
                        case ' ':  case '!':  case '"':  case '#':
                        case '$':  case '%':  case '&':  case '\'':
                        case '(':  case ')':  case '*':  case '+':
                        case ',':  case '-':  case '.':  case '/':
                        case '0':  case '1':  case '2':  case '3':
                        case '4':  case '5':  case '6':  case '7':
                        case '8':  case '9':  case ':':  case ';':
                        case '<':  case '=':  case '>':  case '?':
                        case '@':  case 'A':  case 'B':  case 'C':
                        case 'D':  case 'E':  case 'F':  case 'G':
                        case 'H':  case 'I':  case 'J':  case 'K':
                        case 'L':  case 'M':  case 'N':  case 'O':
                        case 'P':  case 'Q':  case 'R':  case 'S':
                        case 'T':  case 'U':  case 'V':  case 'W':
                        case 'X':  case 'Y':  case 'Z':  case '[':
                        case '\\':  case ']':  case '^':  case '_':
                        case '`':  case 'a':  case 'b':  case 'c':
                        case 'd':  case 'e':  case 'f':  case 'g':
                        case 'h':  case 'i':  case 'j':  case 'k':
                        case 'l':  case 'm':  case 'n':  case 'o':
```

```
                    case 'p':  case 'q':  case 'r':  case 's':
                    case 't':  case 'u':  case 'v':  case 'w':
                    case 'x':  case 'y':  case 'z':  case '{':
                    case '|':  case '}':  case '~':
                    {
                            matchRange(' ','~');
                            break;
                    }
                    case '\t':
                    {
                            match('\t');
                            break;
                    }
                    default:
                    {
                            break _loop33;
                    }
                    }
            } while (true);
            }
            mNL(false);

            _ttype = Token.SKIP;

            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                    _token = makeToken(_ttype);
                    _token.setText(new String(text.getBuffer(), _begin, text.length()-
_begin));
            }
            _returnToken = _token;
    }

    public final void mWS(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = WS;
            int _saveIndex;

            {
            switch ( LA(1)) {
            case ' ':
            {
                    match(' ');
                    break;
            }
            case '\t':
```

```
		{
			match('\t');
			break;
		}
		default:
		{
			throw new NoViableAltForCharException((char)LA(1),
getFilename(), getLine(), getColumn());
		}
		}
		}

		_ttype = Token.SKIP;

		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-
_begin));
		}
		_returnToken = _token;
	}

	public final void mINFILE(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = INFILE;
		int _saveIndex;

		{
		switch ( LA(1)) {
		case 'I':
		{
			match("INFILE");
			break;
		}
		case 'i':
		{
			match("infile");
			break;
		}
		default:
		{
			throw new NoViableAltForCharException((char)LA(1),
getFilename(), getLine(), getColumn());
		}
		}
```

```
		}
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-
_begin));
		}
		_returnToken = _token;
	}

	public final void mOUTFILE(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = OUTFILE;
		int _saveIndex;

		{
		switch ( LA(1)) {
		case 'O':
		{
			match("OUTFILE");
			break;
		}
		case 'o':
		{
			match("outfile");
			break;
		}
		default:
		{
			throw new NoViableAltForCharException((char)LA(1),
getFilename(), getLine(), getColumn());
		}
		}
		}
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-
_begin));
		}
		_returnToken = _token;
	}

	public final void mSstart(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = Sstart;
```

```
		int _saveIndex;

		{
		switch ( LA(1)) {
		case 'S':
		{
			match("START");
			break;
		}
		case 's':
		{
			match("start");
			break;
		}
		default:
		{
			throw new NoViableAltForCharException((char)LA(1),
getFilename(), getLine(), getColumn());
		}
		}
		}
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-
_begin));
		}
		_returnToken = _token;
	}

	public final void mIF(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = IF;
		int _saveIndex;

		{
		switch ( LA(1)) {
		case 'I':
		{
			match("IF");
			break;
		}
		case 'i':
		{
			match("if");
			break;
```

```
		}
		default:
		{
			throw new NoViableAltForCharException((char)LA(1),
getFilename(), getLine(), getColumn());
		}
		}
		}
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-
_begin));
		}
		_returnToken = _token;
	}

	public final void mTHEN(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = THEN;
		int _saveIndex;

		{
		switch ( LA(1)) {
		case 'T':
		{
			match("THEN");
			break;
		}
		case 't':
		{
			match("then");
			break;
		}
		default:
		{
			throw new NoViableAltForCharException((char)LA(1),
getFilename(), getLine(), getColumn());
		}
		}
		}
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-
_begin));
		}
```

```
		_returnToken = _token;
	}

	public final void mFIELD(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = FIELD;
		int _saveIndex;

		{
		switch ( LA(1)) {
		case 'N':
		{
			match("NAME");
			break;
		}
		case 'n':
		{
			match("name");
			break;
		}
		default:
		{
			throw new NoViableAltForCharException((char)LA(1),
getFilename(), getLine(), getColumn());
		}
		}
		}
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-
_begin));
		}
		_returnToken = _token;
	}

	public final void mFW(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = FW;
		int _saveIndex;

		{
		switch ( LA(1)) {
		case 'V':
		{
```

```
                    match("VALUE");
                    break;
                }
                case 'v':
                {
                    match("value");
                    break;
                }
                default:
                {
                    throw new NoViableAltForCharException((char)LA(1),
getFilename(), getLine(), getColumn());
                }
                }
                }
                if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                    _token = makeToken(_ttype);
                    _token.setText(new String(text.getBuffer(), _begin, text.length()-
_begin));
                }
                _returnToken = _token;
        }

        public final void mDELIMITER(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
                int _ttype; Token _token=null; int _begin=text.length();
                _ttype = DELIMITER;
                int _saveIndex;

                {
                switch ( LA(1)) {
                case 'D':
                {
                    match("DELIMITER");
                    break;
                }
                case 'd':
                {
                    match("delimiter");
                    break;
                }
                default:
                {
                    throw new NoViableAltForCharException((char)LA(1),
getFilename(), getLine(), getColumn());
                }
```

```
		}
		}
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-
_begin));
		}
		_returnToken = _token;
	}


	private static final long[] mk_tokenSet_0() {
		long[] data = new long[8];
		data[0]=-17179869192L;
		for (int i = 1; i<=3; i++) { data[i]=-1L; }
		return data;
	}
	public static final BitSet _tokenSet_0 = new BitSet(mk_tokenSet_0());

	}
```

SimpParser.java

```
// $ANTLR 2.7.5 (20050201): "hba.g" -> "SimpParser.java"$

import antlr.TokenBuffer;
import antlr.TokenStreamException;
import antlr.TokenStreamIOException;
import antlr.ANTLRException;
import antlr.LLkParser;
import antlr.Token;
import antlr.TokenStream;
import antlr.RecognitionException;
import antlr.NoViableAltException;
import antlr.MismatchedTokenException;
import antlr.SemanticException;
import antlr.ParserSharedInputState;
import antlr.collections.impl.BitSet;
import antlr.collections.AST;
import java.util.Hashtable;
import antlr.ASTFactory;
import antlr.ASTPair;
import antlr.collections.impl.ASTArray;

public class SimpParser extends antlr.LLkParser       implements
SimpWalkerTokenTypes
```

```
 {

protected SimpParser(TokenBuffer tokenBuf, int k) {
  super(tokenBuf,k);
  tokenNames = _tokenNames;
  buildTokenTypeASTClassMap();
  astFactory = new ASTFactory(getTokenTypeToASTClassMap());
}

public SimpParser(TokenBuffer tokenBuf) {
  this(tokenBuf,2);
}

protected SimpParser(TokenStream lexer, int k) {
  super(lexer,k);
  tokenNames = _tokenNames;
  buildTokenTypeASTClassMap();
  astFactory = new ASTFactory(getTokenTypeToASTClassMap());
}

public SimpParser(TokenStream lexer) {
  this(lexer,2);
}

public SimpParser(ParserSharedInputState state) {
  super(state,2);
  tokenNames = _tokenNames;
  buildTokenTypeASTClassMap();
  astFactory = new ASTFactory(getTokenTypeToASTClassMap());
}

        public final void hba_def() throws RecognitionException, TokenStreamException
{

                returnAST = null;
                ASTPair currentAST = new ASTPair();
                AST hba_def_AST = null;

                try {      // for error handling
                        AST tmp1_AST = null;
                        tmp1_AST = astFactory.create(LT(1));
                        astFactory.makeASTRoot(currentAST, tmp1_AST);
                        match(Sstart);
                        match(SEMI);
                        ifile_stmt();
                        astFactory.addASTChild(currentAST, returnAST);
```

```
                ofile_stmt();
                astFactory.addASTChild(currentAST, returnAST);
                de_stmt();
                astFactory.addASTChild(currentAST, returnAST);
                if_then_stmt();
                astFactory.addASTChild(currentAST, returnAST);
                hba_def_AST = (AST)currentAST.root;
        }
        catch (RecognitionException ex) {
                reportError(ex);
                recover(ex,_tokenSet_0);
        }
        returnAST = hba_def_AST;
    }

    public final void ifile_stmt() throws RecognitionException,
TokenStreamException {

        returnAST = null;
        ASTPair currentAST = new ASTPair();
        AST ifile_stmt_AST = null;

        try {      // for error handling
                AST tmp3_AST = null;
                tmp3_AST = astFactory.create(LT(1));
                astFactory.makeASTRoot(currentAST, tmp3_AST);
                match(INFILE);
                match(ASSIGN);
                AST tmp5_AST = null;
                tmp5_AST = astFactory.create(LT(1));
                astFactory.addASTChild(currentAST, tmp5_AST);
                match(STRING);
                match(SEMI);
                ifile_stmt_AST = (AST)currentAST.root;
        }
        catch (RecognitionException ex) {
                reportError(ex);
                recover(ex,_tokenSet_1);
        }
        returnAST = ifile_stmt_AST;
    }

    public final void ofile_stmt() throws RecognitionException,
TokenStreamException {

        returnAST = null;
```

```
		ASTPair currentAST = new ASTPair();
		AST ofile_stmt_AST = null;

		try {	// for error handling
			AST tmp7_AST = null;
			tmp7_AST = astFactory.create(LT(1));
			astFactory.makeASTRoot(currentAST, tmp7_AST);
			match(OUTFILE);
			match(ASSIGN);
			AST tmp9_AST = null;
			tmp9_AST = astFactory.create(LT(1));
			astFactory.addASTChild(currentAST, tmp9_AST);
			match(STRING);
			match(SEMI);
			ofile_stmt_AST = (AST)currentAST.root;
		}
		catch (RecognitionException ex) {
			reportError(ex);
			recover(ex,_tokenSet_2);
		}
		returnAST = ofile_stmt_AST;
	}

	public final void de_stmt() throws RecognitionException, TokenStreamException
{

		returnAST = null;
		ASTPair currentAST = new ASTPair();
		AST de_stmt_AST = null;

		try {	// for error handling
			AST tmp11_AST = null;
			tmp11_AST = astFactory.create(LT(1));
			astFactory.makeASTRoot(currentAST, tmp11_AST);
			match(DELIMITER);
			match(ASSIGN);
			AST tmp13_AST = null;
			tmp13_AST = astFactory.create(LT(1));
			astFactory.addASTChild(currentAST, tmp13_AST);
			match(STRING);
			match(SEMI);
			de_stmt_AST = (AST)currentAST.root;
		}
		catch (RecognitionException ex) {
			reportError(ex);
			recover(ex,_tokenSet_3);
```

```
		}
		returnAST = de_stmt_AST;
	}

	public final void if_then_stmt() throws RecognitionException,
TokenStreamException {

		returnAST = null;
		ASTPair currentAST = new ASTPair();
		AST if_then_stmt_AST = null;

		try {	// for error handling
			AST tmp15_AST = null;
			tmp15_AST = astFactory.create(LT(1));
			astFactory.makeASTRoot(currentAST, tmp15_AST);
			match(IF);
			match(FIELD);
			match(ASSIGN);
			AST tmp18_AST = null;
			tmp18_AST = astFactory.create(LT(1));
			astFactory.addASTChild(currentAST, tmp18_AST);
			match(STRING);
			match(THEN);
			then_stmt();
			astFactory.addASTChild(currentAST, returnAST);
			match(SEMI);
			if_then_stmt_AST = (AST)currentAST.root;
		}
		catch (RecognitionException ex) {
			reportError(ex);
			recover(ex,_tokenSet_0);
		}
		returnAST = if_then_stmt_AST;
	}

	public final void then_stmt() throws RecognitionException,
TokenStreamException {

		returnAST = null;
		ASTPair currentAST = new ASTPair();
		AST then_stmt_AST = null;

		try {	// for error handling
			AST tmp21_AST = null;
			tmp21_AST = astFactory.create(LT(1));
			astFactory.makeASTRoot(currentAST, tmp21_AST);
```

```
                match(FW);
                match(ASSIGN);
                AST tmp23_AST = null;
                tmp23_AST = astFactory.create(LT(1));
                astFactory.addASTChild(currentAST, tmp23_AST);
                match(STRING);
                then_stmt_AST = (AST)currentAST.root;
        }
        catch (RecognitionException ex) {
                reportError(ex);
                recover(ex,_tokenSet_4);
        }
        returnAST = then_stmt_AST;
}


public static final String[] _tokenNames = {
        "<0>",
        "EOF",
        "<2>",
        "NULL_TREE_LOOKAHEAD",
        "ASSIGN",
        "LPAREN",
        "RPAREN",
        "SEMI",
        "LETTER",
        "DIGIT",
        "NUMBER",
        "STRING",
        "NL",
        "COMMENT",
        "WS",
        "INFILE",
        "OUTFILE",
        "Sstart",
        "IF",
        "THEN",
        "FIELD",
        "FW",
        "DELIMITER"
};

protected void buildTokenTypeASTClassMap() {
        tokenTypeToASTClassMap=null;
};
```

```
	private static final long[] mk_tokenSet_0() {
		long[] data = { 2L, 0L};
		return data;
	}
	public static final BitSet _tokenSet_0 = new BitSet(mk_tokenSet_0());
	private static final long[] mk_tokenSet_1() {
		long[] data = { 65536L, 0L};
		return data;
	}
	public static final BitSet _tokenSet_1 = new BitSet(mk_tokenSet_1());
	private static final long[] mk_tokenSet_2() {
		long[] data = { 4194304L, 0L};
		return data;
	}
	public static final BitSet _tokenSet_2 = new BitSet(mk_tokenSet_2());
	private static final long[] mk_tokenSet_3() {
		long[] data = { 262144L, 0L};
		return data;
	}
	public static final BitSet _tokenSet_3 = new BitSet(mk_tokenSet_3());
	private static final long[] mk_tokenSet_4() {
		long[] data = { 128L, 0L};
		return data;
	}
	public static final BitSet _tokenSet_4 = new BitSet(mk_tokenSet_4());

	}
```

SimpParserTokenTypes.java

```
// $ANTLR 2.7.5 (20050201): "simple.g" -> "SimpParser.java"$

public interface SimpParserTokenTypes {
	int EOF = 1;
	int NULL_TREE_LOOKAHEAD = 3;
	int Sstart = 4;
	int SEMI = 5;
	int INFILE = 6;
	int ASSIGN = 7;
	int ID = 8;
	int OUTFILE = 9;
	int IF = 10;
	int FIELD = 11;
	int STRING = 12;
	int THNEN = 13;
	int FW = 14;
```

```
}
```

SimpWalker.java

```
// $ANTLR 2.7.5 (20050201): "hba.g" -> "SimpWalker.java"$

import antlr.TreeParser;
import antlr.Token;
import antlr.collections.AST;
import antlr.RecognitionException;
import antlr.ANTLRException;
import antlr.NoViableAltException;
import antlr.MismatchedTokenException;
import antlr.SemanticException;
import antlr.collections.impl.BitSet;
import antlr.ASTPair;
import antlr.collections.impl.ASTArray;


public class SimpWalker extends antlr.TreeParser       implements
SimpWalkerTokenTypes
 {

  public static String Fin;
  public static String Fout;
  public static String DeLi;
  public static String Fnm;
  public static String Fvlu;


 public void initialize()
 {

   Fin=null;
   Fout=null;
   DeLi=null;
   Fnm=null;
   Fvlu=null;

 }
public SimpWalker() {
        tokenNames = _tokenNames;
}

        public final String  hba_def(AST _t) throws RecognitionException {
                String s;
```

```
AST hba_def_AST_in = (_t == ASTNULL) ? null : (AST)_t;
AST st = null;
AST m = null;
AST n = null;
AST o = null;
AST q = null;
AST t = null;
AST r = null;
AST fw = null;
AST g = null;
AST f = null;
AST h = null;


s = "a1";

String t1,t2,t3,f1,f2=null;



try {      // for error handling
        if (_t==null) _t=ASTNULL;
        switch ( _t.getType()) {
        case Sstart:
        {
                AST __t2 = _t;
                st = _t==ASTNULL ? null :(AST)_t;
                match(_t,Sstart);
                _t = _t.getFirstChild();
                t1=hba_def(_t);
                _t = _retTree;
                t2=hba_def(_t);
                _t = _retTree;
                t3=hba_def(_t);
                _t = _retTree;
                f1=hba_def(_t);
                _t = _retTree;
                _t = __t2;
                _t = _t.getNextSibling();


                        HBALib.HBAfunc (Fin, Fout, DeLi, Fnm, Fvlu);


                break;
```

```
}
case INFILE:
{
	AST __t3 = _t;
	m = _t==ASTNULL ? null :(AST)_t;
	match(_t,INFILE);
	_t = _t.getFirstChild();
	n = (AST)_t;
	match(_t,STRING);
	_t = _t.getNextSibling();
	_t = __t3;
	_t = _t.getNextSibling();
	Fin = n.getText();

	//	System.out.println( Fin ); System.out.println();


	break;
}
case OUTFILE:
{
	AST __t4 = _t;
	o = _t==ASTNULL ? null :(AST)_t;
	match(_t,OUTFILE);
	_t = _t.getFirstChild();
	q = (AST)_t;
	match(_t,STRING);
	_t = _t.getNextSibling();
	_t = __t4;
	_t = _t.getNextSibling();
	Fout = q.getText();

	//	System.out.println( Fout ); System.out.println();


	break;
}
case DELIMITER:
{
	AST __t5 = _t;
	t = _t==ASTNULL ? null :(AST)_t;
	match(_t,DELIMITER);
	_t = _t.getFirstChild();
	r = (AST)_t;
	match(_t,STRING);
	_t = _t.getNextSibling();
```

```
		_t = __t5;
		_t = _t.getNextSibling();
		DeLi = r.getText();

		//		System.out.println( DeLi ); System.out.println();


		break;
	}
	case FW:
	{
		AST __t6 = _t;
		fw = _t==ASTNULL ? null :(AST)_t;
		match(_t,FW);
		_t = _t.getFirstChild();
		g = (AST)_t;
		match(_t,STRING);
		_t = _t.getNextSibling();
		_t = __t6;
		_t = _t.getNextSibling();
		Fvlu = g.getText();

		//		System.out.println( Fvlu ); System.out.println();


		break;
	}
	case IF:
	{
		AST __t7 = _t;
		f = _t==ASTNULL ? null :(AST)_t;
		match(_t,IF);
		_t = _t.getFirstChild();
		h = (AST)_t;
		match(_t,STRING);
		_t = _t.getNextSibling();
		f2=hba_def(_t);
		_t = _retTree;
		_t = __t7;
		_t = _t.getNextSibling();
		Fnm = h.getText();

		//		System.out.println( Fnm ); System.out.println();


		break;
```

```
                }
                default:
                {
                        throw new NoViableAltException(_t);
                }
                }
        }
        catch (RecognitionException ex) {
                reportError(ex);
                if (_t!=null) {_t = _t.getNextSibling();}
        }
        _retTree = _t;
        return s;
}


public static final String[] _tokenNames = {
        "<0>",
        "EOF",
        "<2>",
        "NULL_TREE_LOOKAHEAD",
        "ASSIGN",
        "LPAREN",
        "RPAREN",
        "SEMI",
        "LETTER",
        "DIGIT",
        "NUMBER",
        "STRING",
        "NL",
        "COMMENT",
        "WS",
        "INFILE",
        "OUTFILE",
        "Sstart",
        "IF",
        "THEN",
        "FIELD",
        "FW",
        "DELIMITER"
};

}
```

SimpWalkerTokenTypes.java

```
// $ANTLR 2.7.5 (20050201): "hba.g" -> "SimpWalker.java"$

public interface SimpWalkerTokenTypes {
	int EOF = 1;
	int NULL_TREE_LOOKAHEAD = 3;
	int ASSIGN = 4;
	int LPAREN = 5;
	int RPAREN = 6;
	int SEMI = 7;
	int LETTER = 8;
	int DIGIT = 9;
	int NUMBER = 10;
	int STRING = 11;
	int NL = 12;
	int COMMENT = 13;
	int WS = 14;
	int INFILE = 15;
	int OUTFILE = 16;
	int Sstart = 17;
	int IF = 18;
	int THEN = 19;
	int FIELD = 20;
	int FW = 21;
	int DELIMITER = 22;
}
```

Bibliography

[1] Christopher Conway, Cheng-Hong Li, Megan Pengelly. Pencil: A Petri Net Specification Language for Java, Columbia University, 2002