TingStim

[Final Report version 1.0]

COMS W4115 [Spring 2006 CVN]

Programming Languages and Translators

http://www1.cs.columbia.edu/~sedwards/classes/2006/w4115-spring/project.html

Prof. Stephen Edwards sedwards@cs.columbia.edu

By Alvin Ting

ating410@yahoo.com

at2337@columbia.edu

2006/05/09

TingStim:

Stimulator / Interface Test Driver

1 Introduction

1.1 Description:

TingStim is a simple, intuitive, and efficient language for stimulating communication interfaces. TingStim is used to test point-to-point terminals such as serial equipment as well as a wide array of networked devices such as routers or autonomous computing nodes. Using the TingStim language, the programmer captures the messages to be sent across the interface of interest along with the expected response messages in a plain text file called a scenario file. The scenario file is then compiled generating the equivalent java source file which in turn is compiled and ran on the ubiquitous java runtime environment. Finally, the results recorded by the interpreter of the interaction between the device of interest and the computer running the interpreter are analyzed by the programmer and compared with the expected results to validate and qualify the behavior of the device.

1.2 Ease of Use

TingStim is simple, easy to learn, and intuitive. There are no complicated constructs or recursive functions. Only basic and necessary features are included in the language in order not to confuse the programmer with too many options. Yet, the TingStim is powerful enough to stimulate even the most complex interfaces.

1.3 Robust

TingStim “scenario files” are robust. For instance, if the communication interface changes from a Serial to an Ethernet connection, the original scenario file can easily be adapted by changing only a few lines in order to use the same message exchange sequence for the new interface.

1.4 Architecture Neutral and Portable

TingStim is an interpreted language and since the interpreter is written in Java, TingStim is architecture neutral. Scenario files written in TingStim are just plain text files making them readable on any text editor. Compiled TingStim scenarios can be easily seamlessly ported to any platform that has Java Virtual Machine (JVM) running on it.

1.5 High Level

TingStim takes care of the complicated communication port initialization, connection, re-connection, etc.; allowing the programmer more time to focus on testing the device and not have to worry about these other mundane details.

1.6 Automation

TingStim scenario files, once compiled and executed by the interpreter, require no further human interaction until the completion of the entire scenario. This allows the operator the ability to run long scenarios overnight.

1.7 Possible functionality or extensions

- Read canned or predefined messages from file so that scenario files can share common messages
- Inclusion of regular expressions in “expect” string matching capabilities
- Display interface messages content graphically
- Support syntax highlighting in custom or COTS editor
- Feature more interface and protocol support – FTP, USB, etc.
- Validate messages automatically

1.8 Related Works

LabVIEW by National Instruments is the industry-leading software tool for designing test, measurement, and control systems. It uses a graphical programming language similar to flowcharting. LabVIEW can be purchased for around $3000.

2 Language Tutorial

2.1 Example Scenario File

```
01   /* example.scn – Tutorial */
02   disp “Initializing Communications...”;
03   open “Ethernet” “128.220.38.52” “9871” “9990”;
04
05   err_val = 0xDEAD;
06   err_msg = “Error reported by device.”;
07
08   disp “Press 1 to Continue”;
09   interact ‘1’ { disp “Ready to begin scenario.”; };
10
11   wait 100; // one hundred milliseconds
12
13   send 0b0001 0b0101 0b1111 0b0101;
14   expect {
15        0xF00D  { send 0xDFF0 0xFFFF; }
16        err_val {
17             // Handle error
18             disp err_msg;
19        }
20   };
21
22   close;
```

2.2 Example Scenario File Explanation

Line 2: Simple display command to output specified literal or identifier to standard output. Display can be used to print out literals (strings, integers, and characters) or an identifier which is a container for a literal. (Reference section 3.4.6)

Line 3: The first string following the open command indicates what type of port should be opened, in this case, an Ethernet port. The remaining string and two decimal values correspond to the server address, send port number, and receive port number. Open must be called before any of the “communication statements” such as send or expect, because these statements rely on a port to be opened prior to use. (Reference section 3.4.8)

Line 5-6: Variables are declared and initialized for use later. Variables must be initialized prior to use. (Reference section 3.4.2)

Line 9: Waits for user to press a key before continuing. If the pressed key does not match the list of expected inputs, the program will issue an alert and loop until that key is pressed. (Reference 3.4.8.3)

Line 11: Pause scenario for indicated number of milliseconds. (Reference 3.4.7)

Line 13: Send binary number sequence to other device via Ethernet socket. The binary number sequence is actually concantenated to form one integer which is converted to decimal form prior to sending. Send can also take a sequence of other integer literals as well as a string literal. (Reference 3.4.8.1)

Line 14: Designated expected responses from the device and the actions to perform for each of those responses. If none of the expected responses is received, then an alert is displayed and the scenario continues. (Reference 3.4.8.2)

Line 22: End of scenario. [Optional] (Reference 3.4.8.5)

Other useful constructs are if/else and a while loop, both of these statements depend upon the evaluation of an equality expression. (Reference 3.4.3 and 3.4.4, respectively)

2.3 Running Example Scenario File

Assuming you have set the necessary classpath and environment variables, you'll also need the tingstim.jar and antlr.jar. The basic idea is you write an communications/test driver program using the constructs defined in the TingStim Language Reference Manual. This scenario file is then compiled using the TingStim compiler and a java source file is generated as a result of this successful compilation. This java source file contains a single threaded application which executes the translated commands of the input scenario file as native java calls. This java source file is compiled with the usual java compiler and ran with the usual java runtime engine. Follow the steps below to build and run a scenario file:

First compile the scenario file to generate the equivalent java source file (named "example.java":

```
C:\> java –jar tingstim.jar example.scn
```

Next compile the generated java file:

```
C:\> javac –classpath tingstim.jar example.java
```

To run the program, simply run it using the java interpreter:

```
C:\> java example
```

For an example of two programs that communicate with one another and are written by the TingStim compiler see Section 6.3.

3 Language Manual

3.1 Lexical Conventions

3.1.1 Line Terminators

Input characters are divided into lines by line terminators. Lines are terminated by the standard ASCII characters `CR`, or `LF`, or `CR LF`. The two characters `CR` immediately followed by `LF` are counted as one line terminator, not two.

LineTerminator:
ASCII `LF` character, also known as "linefeed"
ASCII `CR` character, also known as "carriage-return"
ASCII `CR` character followed by the ASCII `LF` character

InputCharacter:
Is Not *LineTerminator*

3.1.2 Comments

Comments are textual descriptions and are ignored by the compiler. There are two kinds of comments:

/ text */*	Multi-line comment: All the text from the ASCII characters `/*` to the ASCII characters `*/` is ignored.
// text	Single-line comment: All text from the ASCII characters `//` to the end of the line is ignored.

Comment:
MultiLineComment
SingleLineComment

Comments do not nest. The ASCII characters `/*` and `*/` have no special meaning when preceded with the ASCII characters `//`. In turn, the ASCII characters // has no special meaning in comments that begin with `/*`.

3.1.3 White Space

White space is defined as the ASCII space, horizontal tab, form feed characters, and line terminators. White space will be ignored when used to terminate a comment (a line terminator).

WhiteSpace:

ASCII `SP` character, also known as “space”
ASCII `HT` character, also known as “horizontal tab”
ASCII `FF` character, also known as “form feed”
LineTerminator

3.1.4 Tokens

Tokens are subdivided into 4 classifications: identifiers, keywords, literals, separators, and operators. Tokens must be separated by white space.

Token:

Identifier
Keyword
Literals
Separator
Operator

3.1.4.1 Identifiers

An identifier consists of letter, digits and underscores “_”. The first character of an identifier should be a letter or underscore. Upper and lower case letters are considered different.

3.1.4.2 Keywords

The following sequences of characters, formed from ASCII letters, are reserved for use as keywords and cannot be used as identifiers:

```
close      interact   true
catch      if         false
disp       open
else       send
expect     wait
           while
```

3.1.4.3 Literals

Literal:

IntegerLiteral
StringLiteral
BooleanLiteral
CharacterLiteral

3.1.4.3.1 Integer Literal

Integer literals are divided into 4 classifications relative to the base of the number system: decimal, hexadecimal, octal, or binary.

IntegerLiteral:
DecimalLiteral
HexIntegerLiteral
OctalIntegerLiteral
BinaryIntegerLiteral

DecimalLiteral:
DecimalNumeral

HexIntegerLiteral:
ASCII characters 0x followed by *HexNumeral*

OctalIntegerLiteral:
ASCII characters 0o followed by *OctalNumeral*

BinaryIntegerLiteral:
ASCII characters 0b followed by *BinaryNumeral*

DecialNumeral:
(0-9) one or more times

HexNumeral:
(0..9) one or (A-F) one or more times

OctalNumeral:
0..7 one or more times

BinaryNumeral:
(0 | 1) one or more times

For example,
dec: 10000214
hex: 0xDEAD
octal: 0o077
binary: 0b01010101

3.1.4.3.2 String Literal

A string is a sequence of characters enclosed by ASCII double quotes “ ”.

StringLiteral:
" *StringCharacters* "

StringCharacters:
StringCharacter
StringCharacters StringCharacter

StringCharacter:
InputCharacter but not "

3.1.4.3.3 Boolean Literal

The boolean type has two types, represented by the literals true and false, formed from ASCII letters.

BooleanLiteral:
true | false

3.1.4.3.4 Character Literal

A character literal is expressed as a character or an escape sequence, enclosed in ASCII single quotes ' '.

CharacterLiteral:
' SingleCharacter '
' EscapeSequence '

SingleCharacter:
InputCharacter but not ' or \

3.1.4.4 Separators

The following three ASCII characters are separators (punctuators):
`{ } ; / $`

3.1.4.5 Operators

= == !=

The ASCII character = is the assignment operator. Two consecutive ASCII = characters is the equal-to operator. An ASCII character ! followed by the ASCII character = is the not equal-to operator.

3.2 Expressions

Expressions consist of identifiers, literals, equality, and pattern expressions.

3.2.1 Identifier

An identifier itself is a left-value expression. It will be evaluated to some value bounded to this identifier.

3.2.2 Literal

A literal is a right-value expression. It will be evaluated to the literal itself.

3.2.3 Equality

The equality operators are syntactically left-associative (they group left-to-right).

EqualityExpression:
IdentifierExpression == LiteralExpression
IdentifierExpression != LiteralExpression

The == (equal to) and the != (not equal to) operators are used to compare an identifier and a literal.

3.3 Declarations

Variables are implicitly declared upon assignment.

3.4 Statements

Statements are the basic elements of a scenario. A sequence of statements will be executed sequentially, unless flow-control statements indicate otherwise.

Statement:
ExpressionStatement
AssignmentStatement
ConditionalStatement
LoopStatement
NullStatement

DisplayStatement
CommunicationsStatement
WaitStatement

3.4.1 Expression

Expressions may be used as statements by following them with a semicolon.

Syntax:

<*expr*>;

Constraints:

- none

3.4.2 Assignment

The assignment operator = is syntactically right-associative (they group right-to-left). The value of the left operand is set to the value of the right operand.

Syntax:

```
<identifier-expr>  = <literal-expr>;
<identifier-expr1> = <identifier-expr2>;
```

Constraints:

- In all cases, the left operand must be an identifier.
- The right operand may be a literal or a different identifier.

3.4.3 Conditional

Conditionals are done with if (expression) statement else statement. If the equality expression returns true, the first statement is executed, otherwise the second statement is executed.

Syntax:

```
if ( <equality-expr> )
{
      <statement-list>
}
else
{
      <statement-list>
};
```

Constraints:

- Every “if” must have a corresponding “else.”

3.4.4 Loop

The check is done before the execution of the statement, and will be repeated as long as the test expression is true.

Syntax:

```
while ( <equality-expr> )
{
      <statement-list>
};
```

Constraints:

- none

3.4.5 Null

A null statement has no programmatic function and is stated only for completeness.

Syntax:

;

Constraints:

- none

3.4.6 Display

The display statement is used to print the value of an identifier or literal to the standard output.

Syntax:

```
disp <identifier-expr>;
disp <literal-expr>;
```

Constraints:

- none

3.4.7 Wait

The wait statement pauses scenario execution for the designated period of time.

Syntax:

```
wait <time_milliseconds>;
```

where

<time_milliseconds>:

decimal number

Constraints:

- Time is limited to the largest size integer.

3.4.8 Communications

3.4.8.1 Send

The send statement writes the designated literal(s) to the communication output. In the form which takes a list of integer literals, the literals are concatenated to form one numerical value prior to sending. Integer overflow is handled by setting value to the max value. The following are all valid send statements:

```
send 2002;
send 0b0101 0b0010 0b0001 0b0001;   // integer = 21,009
send 0x001A 0xFFFF;                 // integer = 1,769,471
send 0o001 0o123 0o327;             // integer = 304,855
send “Hello, world”;
```

Syntax:

```
send <decimal-literal>;
send <other-integer-literal-list>;
send <string-literal>;
```

where

```
<other-integer-literal-list>:
      HexIntegerLiteral
      OctalIntegerLiteral
      BinaryIntegerLiteral
```

Constraints:

- The subsequent integer literals in the list must be of the same type as the first literal specified.
- The order of list integer literals is such that the first literal contains the most significant bits and the last literal contains the least significant bits.
- The maximum integer value is equivalent to java.lang.Integer.MAX_VALUE.

3.4.8.2 Expect

The expect statement will block on reading from the current communication input. Once an input unit has been read, it will compare the input value with one or a series of valid expressions. When the first valid expression is found, the actions associated with it will be executed and the statement is considered complete.

Syntax:

```
expect <valid-expr> { <action_list> };
```

```
expect
{
      <valid-expr> { <action_list> }
      <valid-expr> { <action_list> }
      …
};
```

where

```
<valid-expr>:
      DecimalLiteral
      <other-integer-literal-list>
      StringLiteral
      CharacterLiteral
      Pattern

<action_list>:
      Send
      Interact
      Wait
      Loop
      Display
      Assignment
      Conditional
```

Constraints:

- Expect statements do not nest.
- <other-integer-literal-list> is the same as that defined in send statement.
- Read timeouts are currently not implemented or handled.

3.4.8.3 Interact

The interact statement is used to enable the operator to alter the flow of the scenario during execution. It will block until the operator enters a specified keystroke. The resulting keystroke is compared with the series of expected character literals. Once a matching literal is found, the associated list of actions is executed and the statement is considered complete. If no matching literal is found then an alert will be displayed and will loop until a valid keystroke is entered.

Syntax:

```
interact CharacterLiteral { <action_list> };

interact
{
      CharacterLiteral { <action_list> }
      CharacterLiteral { <action_list> }
      …
};
```

Constraints:

- <action_list> is same as those in expect statement but does not include another interact statement since interacts do not nest.

3.4.8.4 Open

The open statement opens the specified communication link. Only the first open statement is executed. Any subsequent open statements are invalid and are ignored.

Syntax:

```
open <type_option> <dev-addr> <send-port> <recv-port>;
```

where

<type_option>:
"RS232" (unimplemented)
"Ethernet"
<dev-addr>:
String literal (ex. "128.220.38.43" or "localhost")
<send-port> and <recv-port>:
Decimal literal (ex. 21 or 12904)

Constraints:

- none

3.4.8.5 Close

The close statement closes the current communication link. It is not necessary to include the close statement and is included only for completeness.

Syntax:

```
close;
```

Constraints:

- none

3.5 Scope Rules

3.5.1 Lexical scope

This language is open scope and has only a global namespace.

4 Project Plan

4.1 Project Timeline

Complete…	Planned	Actual	Notes / Deviation Rationale
White Paper	2006/02/07	2006/02/07	On time
LRM	2006/03/02	2006/03/02	On time
Lexer	2006/03/02	2006/03/02	On time
Parser	2006/03/09	2006/03/20	Got sick for two weeks
Walker	2006/03/24	2006/04/09	Relocated from MA to PA, job xfer.
Interpreter	2006/04/10	2006/04/23	Backlog from previous tasks
Tests	2006/04/24	2006/05/03	Catching up
Final Report	2006/05/03	2006/05/06	On time

The TingStim project officially began after the completion of the white paper / proposal. The language reference manual (LRM) defined TingStim grammar and further elaborated on how some of the key features of the language could be implemented using the defined constructs. The LRM naturally drove the development of the Lexer and the Parser. The Lexer development proved to be the easiest task since it was developed almost concurrently with the LRM. The Parser task proved to be more laborious for numerous reasons. The sole and lead developer / project manager (i.e. myself) suffered from flu-like symptoms for a duration of two weeks. Parser development was also impeded by the midterm review and examination. Eventually, the Parser code reached a state in which initial Walker development could occur. Unfortunately, progress was once again disrupted due to my relocation from Massachusetts to Pennsylvannia, along with a job transfer. As with most projects, these difficulties were addressed but not with some losses. Some aspects of the TingStim language were removed, such as, regular expression or pattern recognition. Java provides a regular expression package and it could possibly be easily integrated into Tingstim's "expect" statement, but this feature was sacrificed in order to have enough time for further development and testing of the key features. Relocating also caused reduction of resources. The initial plan was to use java's communications api for serial communications between two computers connected by an RS232 cable. However, one of the computers was inaccessible until it was delivered to my new residence from the storage unit. Meanwhile all development was done on my laptop. As a result, it was easier to implement the Ethernet communication since no physical connections were required and all testing could be completed on the same single machine. Despite all these adversities, all the key features of the TingStim language were implemented.

4.2 Software Development Environment

Software	Notes
JDK 1.5.06	Java Development Kit
Eclipse 3.1	IDE
Antlr 2.7.6	Grammar compiler/ parser and translator generator
Ant 1.6.5	XML based "make" for automated and customized builds
Antlr Studio 1.1.0	Eclipse plugin, syntax highlighting and antlr grammar debugging

Netbeans 5.0	Another IDE, used rarely, but has a nice Print HTML feature.
Java2Html 5.0	Convert source files to nice syntax-highlighted html to include in this report

The latest version of the java development kit was used for TingStim development. I have used Netbeans on prior projects and was pleased with its interface and features, however, I decided to try Eclipse mainly because of the Antlr Studio plugin which made grammar development easier. I first began writing the code within the Eclipse workspace environment and running a dos batch file to issue the sequence of commands to build the appropriate grammar, java source, and test scenarios. After discovering how to integrate the Ant build.xml script I wrote for this project into the Eclipse builder framework, I started using it and builds proved to be cleaner and more flexible. I used a combination of Netbean's print HTML feature and java2html to convert the source files with the nice syntax highlighting that are included in the appendix of this document. In terms of version control, I have used "cvs" in the past, but I recently discovered that no new development would occur with this open source product anymore. Apparently, it is to be replaced by subversion aka"svn". I downloaded the svn install package which turned out to be a couple megabytes in size. Compare this to the cvs executable which is barely half a megabyte. I was against using svn because at a quick glance it did not seem as intuitive to me as cvs and I was also perplexed as to why the install file was so large. Anyway, I opted to not use either versioning system and reverted back to the old fashion procedure of ritualistically backing up all my files periodically on my flash drive. It was fortunate that I did not need to revert to a previous version of a file. A single developer small project such as this one did not suffer as much not having a more official version control system. In hindsight, I probably would have just used cvs anyway or at least created an Ant task which would automatically copy/archive the source files for me.

4.3 Programming Style

4.3.1 Coding Standards

Generally, adhering to a particular coding standard contributes to the successes of collaborative coding projects. Although this is a single developer project, it is still desirable to follow such a coding standard for improved readability.

4.3.2 ANTLR

- Tabs shall be set to four spaces.
- Statements shall not exceed 80 characters, those statements which do must be divided accordingly and placed in separate lines.
- When defining rules for the lexer or parser, the following format shall be used:

```
send_statement
      : KEYWORD_SEND^
```

```
            (     ID
                  | STRLIT
                  | DECLIT
                  | (HEXLIT)+
                  | (OCTLIT)+
                  | (BINLIT)+
            )
      SEMI!
      ;
DIGIT
      : ('0'..'9')
      ;
```

The first line of a rule definition is the rule name. The second line consists of the colon and part of the rule. Judicious use tabs to justify the inner expressions of the rule shall be employed to clearly distinguish blocks. The last line consists of the semicolon and optionally a single line comment indicating the name of the rule – this helps readability for very long rules.

4.3.3 Java

- Tabs shall be set to four spaces.
- Statements shall not exceed 80 characters, those statements which do must be divided accordingly and placed in separate lines.
- Javadoc comments shall be used whenever possible
- The following format shall be used when dealing with blocked statements:

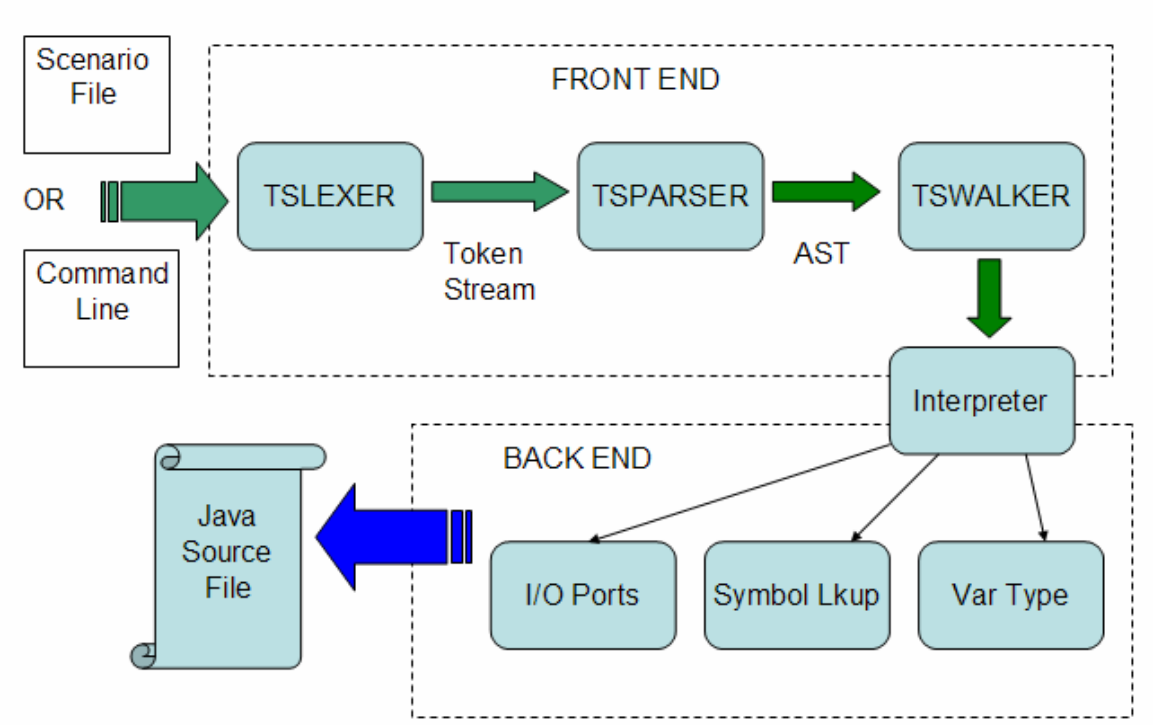
```
      private void writeSCN() {
            try {
                  scnFile.write(programString);
            } catch (IOException ioe) {
                  writeERR("Cannot write scenario file");
            }
      }
```

The left curly brace shall be a single space to the write of the opening statement. The right curly brace shall be justified to match the indentation of the first letter of the opening statement.
- Function names shall begin with a lowercase letter and at least the first letter of subsequent words in the function name shall be capitalized.
- In relational expressions, constants shall be placed to the left of the operator to prevent cases such as "if `(testVariable = 100)`" was written instead of "`if (testVariable == 100)`"

5 Architectural Design

The TingStim interpreter is comprised of: a lexer that reads a plain-text scenario file or input from the command line, a parser that performs syntactical analysis on the input to form an abstract syntax tree, a tree walker that traverses this tree structure to perform semantic analysis, an interpreter which the walker uses to translate scenario calls into

native java code, and a set of backend classes used to perform the business logic of the system. See figure below:



The lexer (TSLexer), the parser (TSParser) and the tree parser (TSWalker) were all written in ANTLR grammar. Both the lexer and the parser are contained in the "TSGrammar.g" file, whereas, the tree parser is contained in the "TSWalker.g" file.

The interpreter (TSInterpreter) contains the translation functions which are called by the tree walker on behalf of each corresponding scenario statement. Error messages are printed for each erroneous or malformed statements detected by the tree parser/intepreter. The interpreter is directly responsible for writing the output java source file.

The backend is comprised of the TSConnectorPort, TSSymbolTable, and TSVariableType classes. The TSConnectorPort contained the logic for opening a communication port with the specified settings, sending a message, receiving a message, and closing that port. The fundamental purpose of the TSConnectorPort class was to encapsulate the various communication connections and the logic involved in doing the above mentioned business tasks. TSConnectorPort would ideally be a composite class containing instances of other communication ports such as Serial or Ethernet. The open statement from the scenario file indicates which of these communication ports whould be realized by the TSConnectorPort. The ability to interchange ports with just a minor change in the scenario file is due in no small part to the power of this design pattern.

The TSSymbolTable is just a little bit more than a wrapper for a hash datastructure. It was created to store values of correctly parsed identifiers as well as retrieve values for comparison with other tokens at run-time. This class is utilized by both the TSInterpreter and generated java source file from the TingStim scenario and thus allowing for resuse of common code.

The TSVariableType is an all purpose variable type used for identifiers. It is the object stored in the TSSymbolTable's hash data structure with its identifier name as the key.

The class called TSMain.java instantiates all the primary classes as well as reads the input file or command line input stream. It is the "Main-class" of the tingstim.jar file created by compiling all the source code and jarring them all together. Details of how this procedure is done can be found in the ANT build script in Section 9.2.

6 Test Plan

6.1 Goal

A series of tests were performed to verify that scenario files were translated into java source files correctly. These tests can be divided into two categories: "Per statement unit tests" and "Integration tests"

6.2 Per Statement Unit Tests

For each statement defined in the language, a corresponding test scenario was created in an attempt to exhaust all possible variations of the statements. This was done to verify that all syntactical and semantic errors were detected. The test scenarios and their generated java source files was included in the Appendix (Section 9.6.1) of this document.

6.3 IntegrationTests

Integration test as the name suggests, attempts to use the statements in various combinations. One pair of integration test is included in the Appendix (Section 9.6.2) of this document. A client and server were created using their source scenario files using the TingStim compiler. The client and server communicate using sockets. This test illustrates that the TingStim compiler does in fact produce viable java code which can communicate over the network and verify received messages.

7 Lessons Learned

New Student – I was originally a student at Johns Hopkins University's Part-time Engineering Graduate Program. PLT was my first CVN course. It was convenient to be able to watch the lectures whenever and wherever but I realized it takes a great deal of discipline to keep true to the schedule in the absence of the classroom environment. In my next CVN course, I would like to use the listserv/email feature more in order to solicit

some of my peers' perspectives on course topics covered and perhaps improve upon my general understanding of the material.

Prepare for the worst / Out of time – As mentioned earlier in Section 4, the month of March was riddled with events which prevented me from making much progress on the project. In the future, I would like to avoid having so much things occurring at the same time. Naturally, some of these events were out of my control, but it would be prudent to set some time in the Project Plan to allow for slips in the schedule or push for a more aggressive schedule and just try to finish earlier.

Read the ANTLR doc – Reading the ANTLR documentation will save you time looking for answers over the web. Most of my questions were answered when I referred to the included ANTLR documentation.

Build scripts save time – ANT is a great tool for managing and customizing builds and related tasks. It is a vast improvement over the often-times-cryptic, recursive Makefiles. I would definitely use it as early as possible in subsequent projects.

Thumbdrives are great – In the absence of a network conection, thumbdrives or flashdrives are great mediums for storing information you need to share between various computer systems. They also make for good backup repositories.

ANTLR is neat – Despite many hours of debugging, ANTLR was a great tool for this class. I was generally pleased with what it could do and how it took away some of the tedious work I would have needed to do in implementing my language. Since ANTLR produced java source code, it made for seamless integration with my java source files.

8 Conclusion

TingStim provides a simple yet powerful alternative for interface testing. TingStim scenario files are easily maintainable and robust. TingStim is a good choice for those who want a quick and efficient way to perform inter-device communication validation.

9 Appendix

9.1 File Listings

```
E:\myData\TingStim\TSE\deploy>jar -tvf tingstim.jar
     0 Mon May 08 23:59:30 EDT 2006 META-INF/
   193 Mon May 08 23:59:28 EDT 2006 META-INF/MANIFEST.MF
     0 Mon May 08 20:38:54 EDT 2006 com/
     0 Mon May 08 20:38:54 EDT 2006 com/tingtech/
     0 Mon May 08 20:38:54 EDT 2006 com/tingtech/tingstim/
 23121 Mon May 08 23:57:06 EDT 2006 TSLexer.java
  4690 Mon May 08 23:59:22 EDT 2006 TSLexer.smap
 32094 Mon May 08 23:59:20 EDT 2006 TSParser.java
  8062 Mon May 08 23:59:20 EDT 2006 TSParser.smap
  1163 Mon May 08 20:38:46 EDT 2006 TSVocabTokenTypes.java
   687 Mon May 08 20:38:46 EDT 2006 TSVocabTokenTypes.txt
 13503 Mon May 08 20:38:48 EDT 2006 TSWalker.java
  3726 Mon May 08 20:38:50 EDT 2006 TSWalker.smap
  1264 Mon May 08 20:38:50 EDT 2006 TSWalkerTokenTypes.java
   754 Mon May 08 20:38:50 EDT 2006 TSWalkerTokenTypes.txt
  2074 Mon May 08 23:59:28 EDT 2006 com/tingtech/tingstim/TSConnectorPort.class
  8377 Mon May 08 23:59:28 EDT 2006 com/tingtech/tingstim/TSInterpreter.class
  9289 Mon May 08 23:59:28 EDT 2006 com/tingtech/tingstim/TSLexer.class
  1668 Mon May 08 23:59:30 EDT 2006 com/tingtech/tingstim/TSMain.class
 10525 Mon May 08 23:59:30 EDT 2006 com/tingtech/tingstim/TSParser.class
  1395 Mon May 08 23:59:28 EDT 2006 com/tingtech/tingstim/TSSymbolTable.class
  1075 Mon May 08 23:59:28 EDT 2006 com/tingtech/tingstim/TSVariableType.class
  1370 Mon May 08 23:59:28 EDT 2006 com/tingtech/tingstim/TSVocabTokenTypes.class
  5734 Mon May 08 23:59:30 EDT 2006 com/tingtech/tingstim/TSWalker.class
  1513 Mon May 08 23:59:28 EDT 2006 com/tingtech/tingstim/TSWalkerTokenTypes.class

E:\myData\TingStim\TSE\deploy>
```

Figure 9.1 Contents of the tingstim jar file

```
E:\myData\TingStim\TSE\com\tingtech\tingstim>dir
 Volume in drive E has no label.
 Volume Serial Number is EFE0-0000

 Directory of E:\myData\TingStim\TSE\com\tingtech\tingstim

04/19/2006  06:56 PM    <DIR>          .
04/19/2006  06:56 PM    <DIR>          ..
05/08/2006  11:59 PM            32,140 TSParser.java
05/08/2006  08:51 PM             1,874 TSMain.java
05/08/2006  11:59 PM            23,135 TSLexer.java
05/08/2006  11:59 PM             1,145 TSVocabTokenTypes.java
05/08/2006  11:59 PM               669 TSVocabTokenTypes.txt
05/08/2006  09:34 PM             3,527 TSWalker.g
05/08/2006  11:58 PM             6,804 TSGrammar.g
05/08/2006  09:34 PM            13,541 TSWalker.java
05/08/2006  09:34 PM             1,246 TSWalkerTokenTypes.java
05/08/2006  09:34 PM               736 TSWalkerTokenTypes.txt
05/08/2006  09:10 PM             3,098 TSSymbolTable.java
05/07/2006  10:42 PM             3,843 TSConnectorPort.java
05/08/2006  11:44 PM            18,686 TSInterpreter.java
05/08/2006  10:29 PM             2,713 TSVariableType.java
              14 File(s)        113,157 bytes
               2 Dir(s)      64,782,336 bytes free
```

Figure 9.2 Source files

```
E:\myData\TingStim\TSE\deploy>dir
 Volume in drive E has no label.
 Volume Serial Number is EFE0-0000

 Directory of E:\myData\TingStim\TSE\deploy

05/08/2006  08:03 PM    <DIR>          .
05/08/2006  08:03 PM    <DIR>          ..
05/08/2006  11:59 PM            36,583 tingstim.jar
05/08/2006  08:03 PM           443,432 antlr.jar
05/08/2006  09:07 PM               571 test_disp.scn
05/08/2006  08:48 PM               898 test_disp.java
05/08/2006  09:51 PM               773 test_assign.scn
05/08/2006  10:20 PM             1,061 test_assign.java
05/08/2006  10:40 PM             1,071 test_if.scn
05/08/2006  10:30 PM             1,847 test_if.java
05/08/2006  10:40 PM               705 test_while.scn
05/08/2006  10:33 PM               944 test_while.java
05/08/2006  10:40 PM               473 test_wait.scn
05/08/2006  10:39 PM               933 test_wait.java
05/08/2006  10:48 PM               597 test_comms_client.scn
05/08/2006  11:04 PM               654 test_comms_server.scn
05/09/2006  12:01 AM               952 test_interact.scn
05/08/2006  11:13 PM             1,599 test_comms_client.java
05/08/2006  11:13 PM             1,858 test_comms_server.java
05/08/2006  11:40 PM               891 test_send.scn
05/08/2006  11:40 PM             1,016 test_send.java
05/09/2006  12:02 AM               966 test_expect.scn
05/09/2006  12:02 AM             1,613 test_expect.java
05/08/2006  11:46 PM               747 test_open.scn
05/08/2006  11:44 PM               721 test_open.java
05/09/2006  12:02 AM             2,366 test_interact.java
              24 File(s)        503,271 bytes
               2 Dir(s)  64,929,792 bytes free
```

Figure 9.3 Scenario files

9.2 Grammar Files

9.2.1 TSGrammar.g

```
header
{
// stuff that is placed at the top of all generated files
package com.tingtech.tingstim;

/*
 * TSGrammar.g: the TingStim lexer and parser in ANTLR grammar
 *
 * @author Alvin Ting - at2337@columbia.edu | ating410@yahoo.com
 *
 */
}


////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
//////////////////////////////// PARSER ////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////


class TSParser extends Parser;
options {
	k=2;
	buildAST = true;
```

```
        exportVocab=TSVocab;
        defaultErrorHandler=false;
}


//////////////////////
//////  STATEMENTS  /////
//////////////////////

tokens {
        STMNT;
        VPAIREX;
        VPAIRIN;
        //CSTMT;
        //VSTMT;
        //CATCHSTMT;
}

progie
        : (primary_statement)+ EOF!
        { #progie = #([STMNT,"PROG"],progie); }
        ;

primary_statement
        : (    expression_statement
        | assignment_statement
        | conditional_statement
        | loop_statement
        | null_statement
        | display_statement
        | wait_statement
        | communications_statement
//      | exit_statement
        ) { #primary_statement = #([STMNT,"STMNT"],primary_statement); }
        ;

compound_statement
        :       LCURLY! ( expression_statement
                        | assignment_statement
                        | conditional_statement
                        | loop_statement
                        | null_statement
                        | display_statement
                        | wait_statement
                        | communications_statement
//                      | exit_statement
                )* RCURLY!
                {#compound_statement = #([STMNT,"CTMNT"], compound_statement); }
        ;

expression_statement
        : primary_expression SEMI!
        ;

assignment_statement
        : identifier_expression ASSIGNEQUAL^
                ( literal_expression
                | identifier_expression
                | pattern_expression
                )
        SEMI!
        ;

conditional_statement
        :       KEYWORD_IF^ LPAREN!
                equality_expression RPAREN!
                compound_statement
                (
                options { //warnWhenFollowAmbig = false;
                                greedy = true;
                }:
                 KEYWORD_ELSE! compound_statement)? SEMI!
```

```
        ;

loop_statement
        : KEYWORD_WHILE^
        LPAREN! equality_expression RPAREN!
        compound_statement
        SEMI!
        ;

null_statement      : SEMI! ; // TODO add a warning

display_statement
        : KEYWORD_DISP^ (identifier_expression | literal_expression) SEMI!
        ;

wait_statement
        : KEYWORD_WAIT^ DECLIT SEMI!
        ;

//exit_statement
//      : KEYWORD_EXIT^ SEMI!
//      ;

communications_statement
        : send_statement
        | expect_statement
        | interact_statement
        | open_statement
        | close_statement
        ;

send_statement
        : KEYWORD_SEND^
                (       ID
                        | STRLIT
                        | DECLIT
                        | (HEXLIT)+
                        | (OCTLIT)+
                        | (BINLIT)+
                )
        SEMI!
        ;

expect_statement
        : KEYWORD_EXPECT^   (
                                valid_pairs_list_expect

                            |       ( LCURLY! (
                                        valid_pairs_list_expect
                                    )+ RCURLY! )
                            )
//          (
//                  options { //warnWhenFollowAmbig = false;
//                                    greedy = true;
//                  }:
//                  catch_statement
//          )?
        SEMI!
        ;

//catch_statement
//      :
//      (KEYWORD_CATCH! valid_stmt_list)
//      { #catch_statement = #([CATCHSTMT,"CATCH"], catch_statement); }
//      ;


valid_pairs_list_expect
        :       (       DECLIT
                        | STRLIT
                        | CHARLIT
                        | PATLIT
```

```
                        | (HEXLIT)+
                        | (OCTLIT)+
                        | (BINLIT)+
                ) valid_stmt_list
        {
                #valid_pairs_list_expect = #([VPAIREX,"VPAIREX"],
                valid_pairs_list_expect);
        }
        ;

valid_stmt_list
        :       LCURLY!         (
                        send_statement
                        | interact_statement
                        | wait_statement
                        | loop_statement
                        | display_statement
                        | assignment_statement
                        | conditional_statement
                )+ RCURLY!
                { #valid_stmt_list = #([STMNT,"VSTMNT"], valid_stmt_list); }
        ;

valid_stmt_list_interact
        :       LCURLY!         (
                        send_statement
                        | wait_statement
                        | loop_statement
                        | display_statement
                        | assignment_statement
                        | conditional_statement
                )+ RCURLY!
                { #valid_stmt_list_interact = #([STMNT,"VSTMNT"],
                                valid_stmt_list_interact); }
        ;


interact_statement
        : KEYWORD_INTERACT^ (
                        (
                                valid_pairs_list_interact
                        )
                |       ( LCURLY! (
                                valid_pairs_list_interact
                        )+ RCURLY! )
                ) SEMI!
        ;

valid_pairs_list_interact
        :       CHARLIT valid_stmt_list_interact
        {
                #valid_pairs_list_interact = #([VPAIRIN,"VPAIRIN"],
                valid_pairs_list_interact);
        }
        ;

open_statement
        : KEYWORD_OPEN^ STRLIT STRLIT DECLIT DECLIT SEMI!
        ;

close_statement
        : KEYWORD_CLOSE^ SEMI!
        ;

////////////////////////
/////  EXPRESSIONS  /////
////////////////////////


primary_expression
        : identifier_expression
        | literal_expression
```

```
        | equality_expression
        | pattern_expression
        ;

equality_expression
        : identifier_expression ((NOTEQUAL^|EQUAL^) literal_expression)
        ;

identifier_expression
        : ID
        ;

literal_expression
        : (STRLIT | CHARLIT | DECLIT | HEXLIT | OCTLIT | BINLIT)
        ;

pattern_expression
        : PATLIT
        ;


class TSLexer extends Lexer;
options
{
    k=2;
       //charVocabulary = '\3'..'\377' | '\u1000'..'\u1fff';
    charVocabulary = '\3'..'\377';
    //charVocabulary='\u0003'..'\uFFFF';
    testLiterals = false;
    //filter=true;
    caseSensitiveLiterals = false;
    exportVocab = TSVocab;
}


// keywords
tokens {
        KEYWORD_CLOSE       = "close";
        KEYWORD_CATCH       = "catch";
        KEYWORD_DISP        = "disp";
        KEYWORD_ELSE        = "else";
        KEYWORD_EXPECT              = "expect";
//      KEYWORD_EXIT        = "exit";
        KEYWORD_IF                  = "if";
        KEYWORD_INTERACT    = "interact";
        KEYWORD_OPEN        = "open";
        KEYWORD_SEND        = "send";
        KEYWORD_WAIT        = "wait";
        KEYWORD_WHILE       = "while";
        KEYWORD_TRUE        = "true";
        KEYWORD_FALSE       = "false";
}


ID
        options { testLiterals = true; }
        : ALPHA ( ALPHA | DIGIT )*
        ; // Identifier

STRLIT
        : '"'! ( ~('"' | '\n' ) | ('"'! '"' ))* '"'!
        ; // String Literal

CHARLIT
        : "\'"! ('\\'('a'..'z') | .) "\'"!
        //{ System.out.println("Got char " + getText()); }
        ; // Character Literal

DECLIT
        : ('0'..'9')+
        ; // Decimal Literal
```

```
HEXLIT
       : '0' 'x' ( '0'..'9' | 'a'..'f' | 'A'..'F' )+
       ; // Hexidecimal Literal

OCTLIT
       : '0' 'o' ('0'..'7')+ // Force use of lowercase 'o' for easier read
       ; // Octal Literal

BINLIT
       : '0' 'b' ('0' | '1')+
       ; // Binary Literal

//BOOLLIT
//     : "true" | "false"
//     ; // Boolean Literal

PATLIT
       : '$'!
       (options {greedy=false;} :
       ~('$' | '\n') )* '$'!
       ; // Pattern

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

NL
       :      ('\n' // UNIX
                     | ('\r' '\n') => '\r' '\n' // DOS
                     | '\r' // MAC
              )
       {
              $setType(Token.SKIP);
              newline(); // LineCounter++
       }
       ; // New Line

COMMENT
       :      ( "/*"
                     (options {greedy=false;} :
                     (NL)
                     | ~( '\n' | '\r' )
                     )* "*/"
              | "//" (~( '\n' | '\r' ))* (NL)
              )
       { $setType(Token.SKIP); }
       ; // Multiple Lines & Single Line Comments

// Operators:
ASSIGNEQUAL     : '=' ;
LPAREN              : '(' ;
RPAREN                      : ')' ;
LCURLY          : '{' ;
RCURLY          : '}' ;
EQUAL               : "==" ;
NOTEQUAL            : "!=" ;

SEMI : ';' ; // Semicolon

protected
ALPHA
       : ('a'..'z'|'A'..'Z' | '_')
       ; // Alpha character

protected
DIGIT
       : ('0'..'9')
       ; // Integer
```

9.2.2 TSWalker.g

```
header
{
// stuff that is placed at the top of all generated files
package com.tingtech.tingstim;

/*
 * TSWalker.g: the TingStim walker in ANTLR grammar
 *
 * @author Alvin Ting - at2337@columbia.edu | ating410@yahoo.com
 *
 */

}


////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
//////////////////////////////  WALKER  ////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////


class TSWalker extends TreeParser;
options {
        importVocab=TSVocab;
}

tokens  {
        // Used for equality checks between two operands
        SYM_NOT_EQUAL;
        SYM_EQUAL;
        SYM_LT_ERROR;
        SYM_RT_ERROR;
}

{
        private TSInterpreter tsi = new TSInterpreter();

        public void configure(String arg) {
                tsi.configure(arg);
        }

        public void exitProgie() {
                tsi.exitMethod("exit");
        }

}

progie returns [ String r ]
        { String a = ""; r=""; }
        :       #(STMNT (stmt:. { r=progie(#stmt);} )* )
                | #(KEYWORD_DISP dstmt:.)       { tsi.dispMethod("disp", #dstmt);
}
                | #(KEYWORD_WAIT wastmt:.)      { tsi.waitMethod("wait",
#wastmt); }
                | #(KEYWORD_SEND sstmt:.)       { tsi.sendMethod("send", #sstmt);
}
                | #(KEYWORD_INTERACT { tsi.interactMethodExprBegin("interact"); }
                                (instmt:.    { r=progie(#instmt); } )*
                                { tsi.interactMethodEnd("interact"); } )
                | #(VPAIRIN
                                r=inleft:progie
                                { tsi.interactMethodExprEnd("interact", #inleft); }
                                (inright:. { r=progie(#inright); } )*
                                { tsi.interactMethodStmtEnd("interact"); } )
```

```
                | #(KEYWORD_WHILE
                                { tsi.whileMethodExprBegin("while"); }
                                r=wleft:progie { tsi.whileMethodExprEnd("while"); }
                                (wright:. { r=progie(#wright); } )*
                                { tsi.whileMethodEnd("while"); } )
                | #(KEYWORD_IF { tsi.ifMethodExprBegin("if"); }
                        r=ileft:progie { tsi.ifMethodExprEnd("if"); }
                        thenp:. (elsep:.)?)
                        {
                                r = progie( #thenp );
                                if ( null != elsep ) {
                                        tsi.elseMethodBegin("else");

                                        r = progie( #elsep );
                                        tsi.elseMethodEnd("else");

                                } else {
                                        tsi.ifMethodNoElse("if");

                                }
                        }

                | #(KEYWORD_EXPECT { tsi.expectMethodExprBegin("expect"); }
                        (vstmt:.        { r=progie(#vstmt); } )* )
                        {       tsi.expectMethodNoCatch("expect");

                                tsi.expectMethodEnd("expect");
                        }
                | #(VPAIREX
                                r=exleft:progie { tsi.expectMethodExprEnd("expect",
#exleft); }
                                (exright:. { r=progie(#exright); } )*
                                { tsi.expectMethodStmtEnd("expect"); } )
                | #(KEYWORD_OPEN ostmt:.)  { tsi.openMethod("open", #ostmt); }
                | #(KEYWORD_CLOSE (a=progie)?)    { System.out.println("close"); }

                | #(ASSIGNEQUAL astmt:.)          { tsi.assignMethod("assign",
#astmt); }
                | #(EQUAL estmt:.)                        { tsi.equalMethod("equal",
#estmt); }
                | #(NOTEQUAL nestmt:.)      { tsi.notEqualMethod("notequal",
#nestmt); }
                | dnum:DECLIT       { System.out.println("num is " +
dnum.getText() ); }
                | hnum:HEXLIT       { System.out.println("num is " +
hnum.getText() ); }
                | onum:OCTLIT       { System.out.println("num is " +
onum.getText() ); }
                | bnum:BINLIT       { System.out.println("num is " +
bnum.getText() ); }
                | str:STRLIT  { System.out.println("str is " + str.getText() ); }

                | chr:CHARLIT{ System.out.println("chr is " + chr.getText() ); }
                | #(id:ID           { System.out.println(" id is " +  id.getText()
); })
//              | KEYWORD_EXIT                                   {
tsi.exitMethod("exit"); }
        ;
```

9.3 Generated Java Source Files

9.3.1 TSParser.java

```
// $ANTLR : "TSGrammar.g" -> "TSParser.java"$

// stuff that is placed at the top of all generated files
package com.tingtech.tingstim;

/*
 * TSGrammar.g: the TingStim lexer and parser in ANTLR grammar
 *
 * @author Alvin Ting - at2337@columbia.edu | ating410@yahoo.com
 *
 */

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 TSParser extends antlr.LLkParser       implements TSVocabT
okenTypes
 {

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

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

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

public TSParser(TokenStream lexer) {
  this(lexer,2);
}
```

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

    public final void progie() throws RecognitionException, TokenStream
Exception {

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

        {
        int _cnt651=0;
        _loop651:
        do {
            if ((_tokenSet_0.member(LA(1)))) {
                primary_statement();
                astFactory.addASTChild(currentAST, returnAST);
            }
            else {
                if ( _cnt651>=1 ) { break _loop651; } else {throw new N
oViableAltException(LT(1), getFilename());}
            }

            _cnt651++;
        } while (true);
        }
        match(Token.EOF_TYPE);
        progie_AST = (AST)currentAST.root;
        progie_AST = (AST)astFactory.make( (new ASTArray(2)).add(astFac
tory.create(STMNT,"PROG")).add(progie_AST));
        currentAST.root = progie_AST;
        currentAST.child = progie_AST!=null &&progie_AST.getFirstChild(
)!=null ?
            progie_AST.getFirstChild() : progie_AST;
        currentAST.advanceChildToEnd();
        progie_AST = (AST)currentAST.root;
        returnAST = progie_AST;
    }

    public final void primary_statement() throws RecognitionException,
TokenStreamException {

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

        {
        switch ( LA(1)) {
        case KEYWORD_IF:
        {
            conditional_statement();
            astFactory.addASTChild(currentAST, returnAST);
            break;
```

```
        }
        case KEYWORD_WHILE:
        {
            loop_statement();
            astFactory.addASTChild(currentAST, returnAST);
            break;
        }
        case SEMI:
        {
            null_statement();
            astFactory.addASTChild(currentAST, returnAST);
            break;
        }
        case KEYWORD_DISP:
        {
            display_statement();
            astFactory.addASTChild(currentAST, returnAST);
            break;
        }
        case KEYWORD_WAIT:
        {
            wait_statement();
            astFactory.addASTChild(currentAST, returnAST);
            break;
        }
        case KEYWORD_SEND:
        case KEYWORD_EXPECT:
        case KEYWORD_INTERACT:
        case KEYWORD_OPEN:
        case KEYWORD_CLOSE:
        {
            communications_statement();
            astFactory.addASTChild(currentAST, returnAST);
            break;
        }
        default:
            if ((_tokenSet_1.member(LA(1))) && (LA(2)==SEMI||LA(2)==NOT
EQUAL||LA(2)==EQUAL)) {
                expression_statement();
                astFactory.addASTChild(currentAST, returnAST);
            }
            else if ((LA(1)==ID) && (LA(2)==ASSIGNEQUAL)) {
                assignment_statement();
                astFactory.addASTChild(currentAST, returnAST);
            }
        else {
            throw new NoViableAltException(LT(1), getFilename());
        }
        }
        }
        primary_statement_AST = (AST)currentAST.root;
        primary_statement_AST = (AST)astFactory.make( (new ASTArray(2))
.add(astFactory.create(STMNT,"STMNT")).add(primary_statement_AST));
        currentAST.root = primary_statement_AST;
        currentAST.child = primary_statement_AST!=null &&primary_statem
ent_AST.getFirstChild()!=null ?
            primary_statement_AST.getFirstChild() : primary_statement_A
```

```
ST;
        currentAST.advanceChildToEnd();
        primary_statement_AST = (AST)currentAST.root;
        returnAST = primary_statement_AST;
    }

    public final void expression_statement() throws RecognitionExceptio
n, TokenStreamException {

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

        primary_expression();
        astFactory.addASTChild(currentAST, returnAST);
        match(SEMI);
        expression_statement_AST = (AST)currentAST.root;
        returnAST = expression_statement_AST;
    }

    public final void assignment_statement() throws RecognitionExceptio
n, TokenStreamException {

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

        identifier_expression();
        astFactory.addASTChild(currentAST, returnAST);
        AST tmp3_AST = null;
        tmp3_AST = astFactory.create(LT(1));
        astFactory.makeASTRoot(currentAST, tmp3_AST);
        match(ASSIGNEQUAL);
        {
        switch ( LA(1)) {
        case DECLIT:
        case STRLIT:
        case HEXLIT:
        case OCTLIT:
        case BINLIT:
        case CHARLIT:
        {
            literal_expression();
            astFactory.addASTChild(currentAST, returnAST);
            break;
        }
        case ID:
        {
            identifier_expression();
            astFactory.addASTChild(currentAST, returnAST);
            break;
        }
        case PATLIT:
        {
            pattern_expression();
            astFactory.addASTChild(currentAST, returnAST);
            break;
```

```
        }
        default:
        {
            throw new NoViableAltException(LT(1), getFilename());
        }
        }
        }
        match(SEMI);
        assignment_statement_AST = (AST)currentAST.root;
        returnAST = assignment_statement_AST;
    }

    public final void conditional_statement() throws RecognitionExcepti
on, TokenStreamException {

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

        AST tmp5_AST = null;
        tmp5_AST = astFactory.create(LT(1));
        astFactory.makeASTRoot(currentAST, tmp5_AST);
        match(KEYWORD_IF);
        match(LPAREN);
        equality_expression();
        astFactory.addASTChild(currentAST, returnAST);
        match(RPAREN);
        compound_statement();
        astFactory.addASTChild(currentAST, returnAST);
        {
        switch ( LA(1)) {
        case KEYWORD_ELSE:
        {
            match(KEYWORD_ELSE);
            compound_statement();
            astFactory.addASTChild(currentAST, returnAST);
            break;
        }
        case SEMI:
        {
            break;
        }
        default:
        {
            throw new NoViableAltException(LT(1), getFilename());
        }
        }
        }
        match(SEMI);
        conditional_statement_AST = (AST)currentAST.root;
        returnAST = conditional_statement_AST;
    }

    public final void loop_statement() throws RecognitionException, Tok
enStreamException {

        returnAST = null;
```

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

        AST tmp10_AST = null;
        tmp10_AST = astFactory.create(LT(1));
        astFactory.makeASTRoot(currentAST, tmp10_AST);
        match(KEYWORD_WHILE);
        match(LPAREN);
        equality_expression();
        astFactory.addASTChild(currentAST, returnAST);
        match(RPAREN);
        compound_statement();
        astFactory.addASTChild(currentAST, returnAST);
        match(SEMI);
        loop_statement_AST = (AST)currentAST.root;
        returnAST = loop_statement_AST;
    }

    public final void null_statement() throws RecognitionException, Tok
enStreamException {

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

        match(SEMI);
        null_statement_AST = (AST)currentAST.root;
        returnAST = null_statement_AST;
    }

    public final void display_statement() throws RecognitionException,
TokenStreamException {

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

        AST tmp15_AST = null;
        tmp15_AST = astFactory.create(LT(1));
        astFactory.makeASTRoot(currentAST, tmp15_AST);
        match(KEYWORD_DISP);
        {
        switch ( LA(1)) {
        case ID:
        {
            identifier_expression();
            astFactory.addASTChild(currentAST, returnAST);
            break;
        }
        case DECLIT:
        case STRLIT:
        case HEXLIT:
        case OCTLIT:
        case BINLIT:
        case CHARLIT:
        {
            literal_expression();
```

```
            astFactory.addASTChild(currentAST, returnAST);
            break;
        }
        default:
        {
            throw new NoViableAltException(LT(1), getFilename());
        }
        }
        }
        match(SEMI);
        display_statement_AST = (AST)currentAST.root;
        returnAST = display_statement_AST;
    }

    public final void wait_statement() throws RecognitionException, Tok
enStreamException {

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

        AST tmp17_AST = null;
        tmp17_AST = astFactory.create(LT(1));
        astFactory.makeASTRoot(currentAST, tmp17_AST);
        match(KEYWORD_WAIT);
        AST tmp18_AST = null;
        tmp18_AST = astFactory.create(LT(1));
        astFactory.addASTChild(currentAST, tmp18_AST);
        match(DECLIT);
        match(SEMI);
        wait_statement_AST = (AST)currentAST.root;
        returnAST = wait_statement_AST;
    }

    public final void communications_statement() throws RecognitionExce
ption, TokenStreamException {

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

        switch ( LA(1)) {
        case KEYWORD_SEND:
        {
            send_statement();
            astFactory.addASTChild(currentAST, returnAST);
            communications_statement_AST = (AST)currentAST.root;
            break;
        }
        case KEYWORD_EXPECT:
        {
            expect_statement();
            astFactory.addASTChild(currentAST, returnAST);
            communications_statement_AST = (AST)currentAST.root;
            break;
        }
        case KEYWORD_INTERACT:
```

```
        {
            interact_statement();
            astFactory.addASTChild(currentAST, returnAST);
            communications_statement_AST = (AST)currentAST.root;
            break;
        }
        case KEYWORD_OPEN:
        {
            open_statement();
            astFactory.addASTChild(currentAST, returnAST);
            communications_statement_AST = (AST)currentAST.root;
            break;
        }
        case KEYWORD_CLOSE:
        {
            close_statement();
            astFactory.addASTChild(currentAST, returnAST);
            communications_statement_AST = (AST)currentAST.root;
            break;
        }
        default:
        {
            throw new NoViableAltException(LT(1), getFilename());
        }
        }
        returnAST = communications_statement_AST;
    }

    public final void compound_statement() throws RecognitionException,
TokenStreamException {

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

        match(LCURLY);
        {
        _loop656:
        do {
            switch ( LA(1)) {
            case KEYWORD_IF:
            {
                conditional_statement();
                astFactory.addASTChild(currentAST, returnAST);
                break;
            }
            case KEYWORD_WHILE:
            {
                loop_statement();
                astFactory.addASTChild(currentAST, returnAST);
                break;
            }
            case SEMI:
            {
                null_statement();
                astFactory.addASTChild(currentAST, returnAST);
                break;
```

```
            }
            case KEYWORD_DISP:
            {
                display_statement();
                astFactory.addASTChild(currentAST, returnAST);
                break;
            }
            case KEYWORD_WAIT:
            {
                wait_statement();
                astFactory.addASTChild(currentAST, returnAST);
                break;
            }
            case KEYWORD_SEND:
            case KEYWORD_EXPECT:
            case KEYWORD_INTERACT:
            case KEYWORD_OPEN:
            case KEYWORD_CLOSE:
            {
                communications_statement();
                astFactory.addASTChild(currentAST, returnAST);
                break;
            }
            default:
                if ((_tokenSet_1.member(LA(1))) && (LA(2)==SEMI||LA(2)=
=NOTEQUAL||LA(2)==EQUAL)) {
                    expression_statement();
                    astFactory.addASTChild(currentAST, returnAST);
                }
                else if ((LA(1)==ID) && (LA(2)==ASSIGNEQUAL)) {
                    assignment_statement();
                    astFactory.addASTChild(currentAST, returnAST);
                }
            else {
                break _loop656;
            }
            }
        } while (true);
        }
        match(RCURLY);
        compound_statement_AST = (AST)currentAST.root;
        compound_statement_AST = (AST)astFactory.make( (new ASTArray(2)
).add(astFactory.create(STMNT,"CTMNT")).add(compound_statement_AST));
        currentAST.root = compound_statement_AST;
        currentAST.child = compound_statement_AST!=null &&compound_stat
ement_AST.getFirstChild()!=null ?
            compound_statement_AST.getFirstChild() : compound_statement
_AST;
        currentAST.advanceChildToEnd();
        compound_statement_AST = (AST)currentAST.root;
        returnAST = compound_statement_AST;
    }

    public final void primary_expression() throws RecognitionException,
 TokenStreamException {

        returnAST = null;
```

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

        switch ( LA(1)) {
        case DECLIT:
        case STRLIT:
        case HEXLIT:
        case OCTLIT:
        case BINLIT:
        case CHARLIT:
        {
            literal_expression();
            astFactory.addASTChild(currentAST, returnAST);
            primary_expression_AST = (AST)currentAST.root;
            break;
        }
        case PATLIT:
        {
            pattern_expression();
            astFactory.addASTChild(currentAST, returnAST);
            primary_expression_AST = (AST)currentAST.root;
            break;
        }
        default:
            if ((LA(1)==ID) && (LA(2)==SEMI)) {
                identifier_expression();
                astFactory.addASTChild(currentAST, returnAST);
                primary_expression_AST = (AST)currentAST.root;
            }
            else if ((LA(1)==ID) && (LA(2)==NOTEQUAL||LA(2)==EQUAL)) {
                equality_expression();
                astFactory.addASTChild(currentAST, returnAST);
                primary_expression_AST = (AST)currentAST.root;
            }
        else {
            throw new NoViableAltException(LT(1), getFilename());
        }
        }
        returnAST = primary_expression_AST;
    }

    public final void identifier_expression() throws RecognitionExcepti
on, TokenStreamException {

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

        AST tmp22_AST = null;
        tmp22_AST = astFactory.create(LT(1));
        astFactory.addASTChild(currentAST, tmp22_AST);
        match(ID);
        identifier_expression_AST = (AST)currentAST.root;
        returnAST = identifier_expression_AST;
    }

    public final void literal_expression() throws RecognitionException,
```

```
TokenStreamException {

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

        {
        switch ( LA(1)) {
        case STRLIT:
        {
            AST tmp23_AST = null;
            tmp23_AST = astFactory.create(LT(1));
            astFactory.addASTChild(currentAST, tmp23_AST);
            match(STRLIT);
            break;
        }
        case CHARLIT:
        {
            AST tmp24_AST = null;
            tmp24_AST = astFactory.create(LT(1));
            astFactory.addASTChild(currentAST, tmp24_AST);
            match(CHARLIT);
            break;
        }
        case DECLIT:
        {
            AST tmp25_AST = null;
            tmp25_AST = astFactory.create(LT(1));
            astFactory.addASTChild(currentAST, tmp25_AST);
            match(DECLIT);
            break;
        }
        case HEXLIT:
        {
            AST tmp26_AST = null;
            tmp26_AST = astFactory.create(LT(1));
            astFactory.addASTChild(currentAST, tmp26_AST);
            match(HEXLIT);
            break;
        }
        case OCTLIT:
        {
            AST tmp27_AST = null;
            tmp27_AST = astFactory.create(LT(1));
            astFactory.addASTChild(currentAST, tmp27_AST);
            match(OCTLIT);
            break;
        }
        case BINLIT:
        {
            AST tmp28_AST = null;
            tmp28_AST = astFactory.create(LT(1));
            astFactory.addASTChild(currentAST, tmp28_AST);
            match(BINLIT);
            break;
        }
        default:
```

```
        {
            throw new NoViableAltException(LT(1), getFilename());
        }
        }
        }
        literal_expression_AST = (AST)currentAST.root;
        returnAST = literal_expression_AST;
    }

    public final void pattern_expression() throws RecognitionException,
 TokenStreamException {

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

        AST tmp29_AST = null;
        tmp29_AST = astFactory.create(LT(1));
        astFactory.addASTChild(currentAST, tmp29_AST);
        match(PATLIT);
        pattern_expression_AST = (AST)currentAST.root;
        returnAST = pattern_expression_AST;
    }

    public final void equality_expression() throws RecognitionException
, TokenStreamException {

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

        identifier_expression();
        astFactory.addASTChild(currentAST, returnAST);
        {
        {
        switch ( LA(1)) {
        case NOTEQUAL:
        {
            AST tmp30_AST = null;
            tmp30_AST = astFactory.create(LT(1));
            astFactory.makeASTRoot(currentAST, tmp30_AST);
            match(NOTEQUAL);
            break;
        }
        case EQUAL:
        {
            AST tmp31_AST = null;
            tmp31_AST = astFactory.create(LT(1));
            astFactory.makeASTRoot(currentAST, tmp31_AST);
            match(EQUAL);
            break;
        }
        default:
        {
            throw new NoViableAltException(LT(1), getFilename());
        }
        }
```

```
        }
        literal_expression();
        astFactory.addASTChild(currentAST, returnAST);
        }
        equality_expression_AST = (AST)currentAST.root;
        returnAST = equality_expression_AST;
    }

    public final void send_statement() throws RecognitionException, Tok
enStreamException {

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

        AST tmp32_AST = null;
        tmp32_AST = astFactory.create(LT(1));
        astFactory.makeASTRoot(currentAST, tmp32_AST);
        match(KEYWORD_SEND);
        {
        switch ( LA(1)) {
        case ID:
        {
            AST tmp33_AST = null;
            tmp33_AST = astFactory.create(LT(1));
            astFactory.addASTChild(currentAST, tmp33_AST);
            match(ID);
            break;
        }
        case STRLIT:
        {
            AST tmp34_AST = null;
            tmp34_AST = astFactory.create(LT(1));
            astFactory.addASTChild(currentAST, tmp34_AST);
            match(STRLIT);
            break;
        }
        case DECLIT:
        {
            AST tmp35_AST = null;
            tmp35_AST = astFactory.create(LT(1));
            astFactory.addASTChild(currentAST, tmp35_AST);
            match(DECLIT);
            break;
        }
        case HEXLIT:
        {
            {
            int _cnt671=0;
            _loop671:
            do {
                if ((LA(1)==HEXLIT)) {
                    AST tmp36_AST = null;
                    tmp36_AST = astFactory.create(LT(1));
                    astFactory.addASTChild(currentAST, tmp36_AST);
                    match(HEXLIT);
                }
```

```
                else {
                    if ( _cnt671>=1 ) { break _loop671; } else {throw n
ew NoViableAltException(LT(1), getFilename());}
                }

                _cnt671++;
            } while (true);
            }
            break;
        }
        case OCTLIT:
        {
            {
            int _cnt673=0;
            _loop673:
            do {
                if ((LA(1)==OCTLIT)) {
                    AST tmp37_AST = null;
                    tmp37_AST = astFactory.create(LT(1));
                    astFactory.addASTChild(currentAST, tmp37_AST);
                    match(OCTLIT);
                }
                else {
                    if ( _cnt673>=1 ) { break _loop673; } else {throw n
ew NoViableAltException(LT(1), getFilename());}
                }

                _cnt673++;
            } while (true);
            }
            break;
        }
        case BINLIT:
        {
            {
            int _cnt675=0;
            _loop675:
            do {
                if ((LA(1)==BINLIT)) {
                    AST tmp38_AST = null;
                    tmp38_AST = astFactory.create(LT(1));
                    astFactory.addASTChild(currentAST, tmp38_AST);
                    match(BINLIT);
                }
                else {
                    if ( _cnt675>=1 ) { break _loop675; } else {throw n
ew NoViableAltException(LT(1), getFilename());}
                }

                _cnt675++;
            } while (true);
            }
            break;
        }
        default:
        {
            throw new NoViableAltException(LT(1), getFilename());
```

```
        }
        }
        }
        match(SEMI);
        send_statement_AST = (AST)currentAST.root;
        returnAST = send_statement_AST;
    }

    public final void expect_statement() throws RecognitionException, T
okenStreamException {

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

        AST tmp40_AST = null;
        tmp40_AST = astFactory.create(LT(1));
        astFactory.makeASTRoot(currentAST, tmp40_AST);
        match(KEYWORD_EXPECT);
        {
        switch ( LA(1)) {
        case DECLIT:
        case STRLIT:
        case HEXLIT:
        case OCTLIT:
        case BINLIT:
        case CHARLIT:
        case PATLIT:
        {
            valid_pairs_list_expect();
            astFactory.addASTChild(currentAST, returnAST);
            break;
        }
        case LCURLY:
        {
            {
            match(LCURLY);
            {
            int _cnt680=0;
            _loop680:
            do {
                if ((_tokenSet_2.member(LA(1)))) {
                    valid_pairs_list_expect();
                    astFactory.addASTChild(currentAST, returnAST);
                }
                else {
                    if ( _cnt680>=1 ) { break _loop680; } else {throw n
ew NoViableAltException(LT(1), getFilename());}
                }

                _cnt680++;
            } while (true);
            }
            match(RCURLY);
            }
            break;
        }
```

```
        default:
        {
            throw new NoViableAltException(LT(1), getFilename());
        }
        }
        }
        match(SEMI);
        expect_statement_AST = (AST)currentAST.root;
        returnAST = expect_statement_AST;
    }

    public final void interact_statement() throws RecognitionException,
 TokenStreamException {

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

        AST tmp44_AST = null;
        tmp44_AST = astFactory.create(LT(1));
        astFactory.makeASTRoot(currentAST, tmp44_AST);
        match(KEYWORD_INTERACT);
        {
        switch ( LA(1)) {
        case CHARLIT:
        {
            {
            valid_pairs_list_interact();
            astFactory.addASTChild(currentAST, returnAST);
            }
            break;
        }
        case LCURLY:
        {
            {
            match(LCURLY);
            {
            int _cnt700=0;
            _loop700:
            do {
                if ((LA(1)==CHARLIT)) {
                    valid_pairs_list_interact();
                    astFactory.addASTChild(currentAST, returnAST);
                }
                else {
                    if ( _cnt700>=1 ) { break _loop700; } else {throw n
ew NoViableAltException(LT(1), getFilename());}
                }

                _cnt700++;
            } while (true);
            }
            match(RCURLY);
            }
            break;
        }
        default:
```

```
        {
            throw new NoViableAltException(LT(1), getFilename());
        }
        }
        }
        match(SEMI);
        interact_statement_AST = (AST)currentAST.root;
        returnAST = interact_statement_AST;
    }

    public final void open_statement() throws RecognitionException, Tok
enStreamException {

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

        AST tmp48_AST = null;
        tmp48_AST = astFactory.create(LT(1));
        astFactory.makeASTRoot(currentAST, tmp48_AST);
        match(KEYWORD_OPEN);
        AST tmp49_AST = null;
        tmp49_AST = astFactory.create(LT(1));
        astFactory.addASTChild(currentAST, tmp49_AST);
        match(STRLIT);
        AST tmp50_AST = null;
        tmp50_AST = astFactory.create(LT(1));
        astFactory.addASTChild(currentAST, tmp50_AST);
        match(STRLIT);
        AST tmp51_AST = null;
        tmp51_AST = astFactory.create(LT(1));
        astFactory.addASTChild(currentAST, tmp51_AST);
        match(DECLIT);
        AST tmp52_AST = null;
        tmp52_AST = astFactory.create(LT(1));
        astFactory.addASTChild(currentAST, tmp52_AST);
        match(DECLIT);
        match(SEMI);
        open_statement_AST = (AST)currentAST.root;
        returnAST = open_statement_AST;
    }

    public final void close_statement() throws RecognitionException, To
kenStreamException {

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

        AST tmp54_AST = null;
        tmp54_AST = astFactory.create(LT(1));
        astFactory.makeASTRoot(currentAST, tmp54_AST);
        match(KEYWORD_CLOSE);
        match(SEMI);
        close_statement_AST = (AST)currentAST.root;
        returnAST = close_statement_AST;
    }
```

```
    public final void valid_pairs_list_expect() throws RecognitionExcep
tion, TokenStreamException {

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

        {
        switch ( LA(1)) {
        case DECLIT:
        {
            AST tmp56_AST = null;
            tmp56_AST = astFactory.create(LT(1));
            astFactory.addASTChild(currentAST, tmp56_AST);
            match(DECLIT);
            break;
        }
        case STRLIT:
        {
            AST tmp57_AST = null;
            tmp57_AST = astFactory.create(LT(1));
            astFactory.addASTChild(currentAST, tmp57_AST);
            match(STRLIT);
            break;
        }
        case CHARLIT:
        {
            AST tmp58_AST = null;
            tmp58_AST = astFactory.create(LT(1));
            astFactory.addASTChild(currentAST, tmp58_AST);
            match(CHARLIT);
            break;
        }
        case PATLIT:
        {
            AST tmp59_AST = null;
            tmp59_AST = astFactory.create(LT(1));
            astFactory.addASTChild(currentAST, tmp59_AST);
            match(PATLIT);
            break;
        }
        case HEXLIT:
        {
            {
            int _cnt684=0;
            _loop684:
            do {
                if ((LA(1)==HEXLIT)) {
                    AST tmp60_AST = null;
                    tmp60_AST = astFactory.create(LT(1));
                    astFactory.addASTChild(currentAST, tmp60_AST);
                    match(HEXLIT);
                }
                else {
                    if ( _cnt684>=1 ) { break _loop684; } else {throw n
ew NoViableAltException(LT(1), getFilename());}
```

```
                }

                _cnt684++;
            } while (true);
            }
            break;
        }
        case OCTLIT:
        {
            {
            int _cnt686=0;
            _loop686:
            do {
                if ((LA(1)==OCTLIT)) {
                    AST tmp61_AST = null;
                    tmp61_AST = astFactory.create(LT(1));
                    astFactory.addASTChild(currentAST, tmp61_AST);
                    match(OCTLIT);
                }
                else {
                    if ( _cnt686>=1 ) { break _loop686; } else {throw n
ew NoViableAltException(LT(1), getFilename());}
                }

                _cnt686++;
            } while (true);
            }
            break;
        }
        case BINLIT:
        {
            {
            int _cnt688=0;
            _loop688:
            do {
                if ((LA(1)==BINLIT)) {
                    AST tmp62_AST = null;
                    tmp62_AST = astFactory.create(LT(1));
                    astFactory.addASTChild(currentAST, tmp62_AST);
                    match(BINLIT);
                }
                else {
                    if ( _cnt688>=1 ) { break _loop688; } else {throw n
ew NoViableAltException(LT(1), getFilename());}
                }

                _cnt688++;
            } while (true);
            }
            break;
        }
        default:
        {
            throw new NoViableAltException(LT(1), getFilename());
        }
        }
        }
```

```
        valid_stmt_list();
        astFactory.addASTChild(currentAST, returnAST);
        valid_pairs_list_expect_AST = (AST)currentAST.root;

                valid_pairs_list_expect_AST = (AST)astFactory.make( (ne
w ASTArray(2)).add(astFactory.create(VPAIREX,"VPAIREX")).add(valid_pair
s_list_expect_AST));

        currentAST.root = valid_pairs_list_expect_AST;
        currentAST.child = valid_pairs_list_expect_AST!=null &&valid_pa
irs_list_expect_AST.getFirstChild()!=null ?
            valid_pairs_list_expect_AST.getFirstChild() : valid_pairs_l
ist_expect_AST;
        currentAST.advanceChildToEnd();
        valid_pairs_list_expect_AST = (AST)currentAST.root;
        returnAST = valid_pairs_list_expect_AST;
    }

    public final void valid_stmt_list() throws RecognitionException, To
kenStreamException {

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

        match(LCURLY);
        {
        int _cnt691=0;
        _loop691:
        do {
            switch ( LA(1)) {
            case KEYWORD_SEND:
            {
                send_statement();
                astFactory.addASTChild(currentAST, returnAST);
                break;
            }
            case KEYWORD_INTERACT:
            {
                interact_statement();
                astFactory.addASTChild(currentAST, returnAST);
                break;
            }
            case KEYWORD_WAIT:
            {
                wait_statement();
                astFactory.addASTChild(currentAST, returnAST);
                break;
            }
            case KEYWORD_WHILE:
            {
                loop_statement();
                astFactory.addASTChild(currentAST, returnAST);
                break;
            }
            case KEYWORD_DISP:
            {
```

```
                display_statement();
                astFactory.addASTChild(currentAST, returnAST);
                break;
            }
            case ID:
            {
                assignment_statement();
                astFactory.addASTChild(currentAST, returnAST);
                break;
            }
            case KEYWORD_IF:
            {
                conditional_statement();
                astFactory.addASTChild(currentAST, returnAST);
                break;
            }
            default:
            {
                if ( _cnt691>=1 ) { break _loop691; } else {throw new N
oViableAltException(LT(1), getFilename());}
            }
            }
            _cnt691++;
        } while (true);
        }
        match(RCURLY);
        valid_stmt_list_AST = (AST)currentAST.root;
        valid_stmt_list_AST = (AST)astFactory.make( (new ASTArray(2)).a
dd(astFactory.create(STMNT,"VSTMNT")).add(valid_stmt_list_AST));
        currentAST.root = valid_stmt_list_AST;
        currentAST.child = valid_stmt_list_AST!=null &&valid_stmt_list_
AST.getFirstChild()!=null ?
            valid_stmt_list_AST.getFirstChild() : valid_stmt_list_AST;
        currentAST.advanceChildToEnd();
        valid_stmt_list_AST = (AST)currentAST.root;
        returnAST = valid_stmt_list_AST;
    }

    public final void valid_stmt_list_interact() throws RecognitionExce
ption, TokenStreamException {

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

        match(LCURLY);
        {
        int _cnt694=0;
        _loop694:
        do {
            switch ( LA(1)) {
            case KEYWORD_SEND:
            {
                send_statement();
                astFactory.addASTChild(currentAST, returnAST);
                break;
            }
```

```
            case KEYWORD_WAIT:
            {
                wait_statement();
                astFactory.addASTChild(currentAST, returnAST);
                break;
            }
            case KEYWORD_WHILE:
            {
                loop_statement();
                astFactory.addASTChild(currentAST, returnAST);
                break;
            }
            case KEYWORD_DISP:
            {
                display_statement();
                astFactory.addASTChild(currentAST, returnAST);
                break;
            }
            case ID:
            {
                assignment_statement();
                astFactory.addASTChild(currentAST, returnAST);
                break;
            }
            case KEYWORD_IF:
            {
                conditional_statement();
                astFactory.addASTChild(currentAST, returnAST);
                break;
            }
            default:
            {
                if ( _cnt694>=1 ) { break _loop694; } else {throw new N
oViableAltException(LT(1), getFilename());}
            }
            }
            _cnt694++;
        } while (true);
        }
        match(RCURLY);
        valid_stmt_list_interact_AST = (AST)currentAST.root;
        valid_stmt_list_interact_AST = (AST)astFactory.make( (new ASTAr
ray(2)).add(astFactory.create(STMNT,"VSTMNT")).add(valid_stmt_list_inte
ract_AST));
        currentAST.root = valid_stmt_list_interact_AST;
        currentAST.child = valid_stmt_list_interact_AST!=null &&valid_s
tmt_list_interact_AST.getFirstChild()!=null ?
            valid_stmt_list_interact_AST.getFirstChild() : valid_stmt_l
ist_interact_AST;
        currentAST.advanceChildToEnd();
        valid_stmt_list_interact_AST = (AST)currentAST.root;
        returnAST = valid_stmt_list_interact_AST;
    }

    public final void valid_pairs_list_interact() throws RecognitionExc
eption, TokenStreamException {
```

```
        returnAST = null;
        ASTPair currentAST = new ASTPair();
        AST valid_pairs_list_interact_AST = null;

        AST tmp67_AST = null;
        tmp67_AST = astFactory.create(LT(1));
        astFactory.addASTChild(currentAST, tmp67_AST);
        match(CHARLIT);
        valid_stmt_list_interact();
        astFactory.addASTChild(currentAST, returnAST);
        valid_pairs_list_interact_AST = (AST)currentAST.root;

                valid_pairs_list_interact_AST = (AST)astFactory.make( (
new ASTArray(2)).add(astFactory.create(VPAIRIN,"VPAIRIN")).add(valid_pa
irs_list_interact_AST));

        currentAST.root = valid_pairs_list_interact_AST;
        currentAST.child = valid_pairs_list_interact_AST!=null &&valid_
pairs_list_interact_AST.getFirstChild()!=null ?
            valid_pairs_list_interact_AST.getFirstChild() : valid_pairs
_list_interact_AST;
        currentAST.advanceChildToEnd();
        valid_pairs_list_interact_AST = (AST)currentAST.root;
        returnAST = valid_pairs_list_interact_AST;
    }


    public static final String[] _tokenNames = {
        "<0>",
        "EOF",
        "<2>",
        "NULL_TREE_LOOKAHEAD",
        "STMNT",
        "VPAIREX",
        "VPAIRIN",
        "LCURLY",
        "RCURLY",
        "SEMI",
        "ASSIGNEQUAL",
        "\"if\"",
        "LPAREN",
        "RPAREN",
        "\"else\"",
        "\"while\"",
        "\"disp\"",
        "\"wait\"",
        "DECLIT",
        "\"send\"",
        "ID",
        "STRLIT",
        "HEXLIT",
        "OCTLIT",
        "BINLIT",
        "\"expect\"",
        "CHARLIT",
        "PATLIT",
        "\"interact\"",
```

```
        "\"open\"",
        "\"close\"",
        "NOTEQUAL",
        "EQUAL",
        "\"catch\"",
        "\"true\"",
        "\"false\"",
        "WS",
        "NL",
        "COMMENT",
        "ALPHA",
        "DIGIT"
    };

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

    private static final long[] mk_tokenSet_0() {
        long[] data = { 2147453440L, 0L};
        return data;
    }
    public static final BitSet _tokenSet_0 = new BitSet(mk_tokenSet_0()
);
    private static final long[] mk_tokenSet_1() {
        long[] data = { 234094592L, 0L};
        return data;
    }
    public static final BitSet _tokenSet_1 = new BitSet(mk_tokenSet_1()
);
    private static final long[] mk_tokenSet_2() {
        long[] data = { 233046016L, 0L};
        return data;
    }
    public static final BitSet _tokenSet_2 = new BitSet(mk_tokenSet_2()
);

    }
```

9.3.2 TSLexer.java

```
// $ANTLR : "TSGrammar.g" -> "TSParser.java"$

// stuff that is placed at the top of all generated files
package com.tingtech.tingstim;

/*
 * TSGrammar.g: the TingStim lexer and parser in ANTLR grammar
 *
 * @author Alvin Ting - at2337@columbia.edu | ating410@yahoo.com
 *
 */
```

```
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 TSParser extends antlr.LLkParser       implements TSVocabT
okenTypes
 {

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

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

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

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

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

    public final void progie() throws RecognitionException, TokenStream
Exception {

        returnAST = null;
        ASTPair currentAST = new ASTPair();
```

```
        AST progie_AST = null;

        {
        int _cnt651=0;
        _loop651:
        do {
            if ((_tokenSet_0.member(LA(1)))) {
                primary_statement();
                astFactory.addASTChild(currentAST, returnAST);
            }
            else {
                if ( _cnt651>=1 ) { break _loop651; } else {throw new N
oViableAltException(LT(1), getFilename());}
            }

            _cnt651++;
        } while (true);
        }
        match(Token.EOF_TYPE);
        progie_AST = (AST)currentAST.root;
        progie_AST = (AST)astFactory.make( (new ASTArray(2)).add(astFac
tory.create(STMNT,"PROG")).add(progie_AST));
        currentAST.root = progie_AST;
        currentAST.child = progie_AST!=null &&progie_AST.getFirstChild(
)!=null ?
            progie_AST.getFirstChild() : progie_AST;
        currentAST.advanceChildToEnd();
        progie_AST = (AST)currentAST.root;
        returnAST = progie_AST;
    }

    public final void primary_statement() throws RecognitionException,
TokenStreamException {

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

        {
        switch ( LA(1)) {
        case KEYWORD_IF:
        {
            conditional_statement();
            astFactory.addASTChild(currentAST, returnAST);
            break;
        }
        case KEYWORD_WHILE:
        {
            loop_statement();
            astFactory.addASTChild(currentAST, returnAST);
            break;
        }
        case SEMI:
        {
            null_statement();
            astFactory.addASTChild(currentAST, returnAST);
            break;
```

```
        }
        case KEYWORD_DISP:
        {
            display_statement();
            astFactory.addASTChild(currentAST, returnAST);
            break;
        }
        case KEYWORD_WAIT:
        {
            wait_statement();
            astFactory.addASTChild(currentAST, returnAST);
            break;
        }
        case KEYWORD_SEND:
        case KEYWORD_EXPECT:
        case KEYWORD_INTERACT:
        case KEYWORD_OPEN:
        case KEYWORD_CLOSE:
        {
            communications_statement();
            astFactory.addASTChild(currentAST, returnAST);
            break;
        }
        default:
            if ((_tokenSet_1.member(LA(1))) && (LA(2)==SEMI||LA(2)==NOT
EQUAL||LA(2)==EQUAL)) {
                expression_statement();
                astFactory.addASTChild(currentAST, returnAST);
            }
            else if ((LA(1)==ID) && (LA(2)==ASSIGNEQUAL)) {
                assignment_statement();
                astFactory.addASTChild(currentAST, returnAST);
            }
        else {
            throw new NoViableAltException(LT(1), getFilename());
        }
        }
        }
        primary_statement_AST = (AST)currentAST.root;
        primary_statement_AST = (AST)astFactory.make( (new ASTArray(2))
.add(astFactory.create(STMNT,"STMNT")).add(primary_statement_AST));
        currentAST.root = primary_statement_AST;
        currentAST.child = primary_statement_AST!=null &&primary_statem
ent_AST.getFirstChild()!=null ?
            primary_statement_AST.getFirstChild() : primary_statement_A
ST;
        currentAST.advanceChildToEnd();
        primary_statement_AST = (AST)currentAST.root;
        returnAST = primary_statement_AST;
    }

    public final void expression_statement() throws RecognitionExceptio
n, TokenStreamException {

        returnAST = null;
        ASTPair currentAST = new ASTPair();
        AST expression_statement_AST = null;
```

```
        primary_expression();
        astFactory.addASTChild(currentAST, returnAST);
        match(SEMI);
        expression_statement_AST = (AST)currentAST.root;
        returnAST = expression_statement_AST;
    }

    public final void assignment_statement() throws RecognitionExceptio
n, TokenStreamException {

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

        identifier_expression();
        astFactory.addASTChild(currentAST, returnAST);
        AST tmp3_AST = null;
        tmp3_AST = astFactory.create(LT(1));
        astFactory.makeASTRoot(currentAST, tmp3_AST);
        match(ASSIGNEQUAL);
        {
        switch ( LA(1)) {
        case DECLIT:
        case STRLIT:
        case HEXLIT:
        case OCTLIT:
        case BINLIT:
        case CHARLIT:
        {
            literal_expression();
            astFactory.addASTChild(currentAST, returnAST);
            break;
        }
        case ID:
        {
            identifier_expression();
            astFactory.addASTChild(currentAST, returnAST);
            break;
        }
        case PATLIT:
        {
            pattern_expression();
            astFactory.addASTChild(currentAST, returnAST);
            break;
        }
        default:
        {
            throw new NoViableAltException(LT(1), getFilename());
        }
        }
        }
        match(SEMI);
        assignment_statement_AST = (AST)currentAST.root;
        returnAST = assignment_statement_AST;
    }
```

```
    public final void conditional_statement() throws RecognitionExcepti
on, TokenStreamException {

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

        AST tmp5_AST = null;
        tmp5_AST = astFactory.create(LT(1));
        astFactory.makeASTRoot(currentAST, tmp5_AST);
        match(KEYWORD_IF);
        match(LPAREN);
        equality_expression();
        astFactory.addASTChild(currentAST, returnAST);
        match(RPAREN);
        compound_statement();
        astFactory.addASTChild(currentAST, returnAST);
        {
        switch ( LA(1)) {
        case KEYWORD_ELSE:
        {
            match(KEYWORD_ELSE);
            compound_statement();
            astFactory.addASTChild(currentAST, returnAST);
            break;
        }
        case SEMI:
        {
            break;
        }
        default:
        {
            throw new NoViableAltException(LT(1), getFilename());
        }
        }
        }
        match(SEMI);
        conditional_statement_AST = (AST)currentAST.root;
        returnAST = conditional_statement_AST;
    }

    public final void loop_statement() throws RecognitionException, Tok
enStreamException {

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

        AST tmp10_AST = null;
        tmp10_AST = astFactory.create(LT(1));
        astFactory.makeASTRoot(currentAST, tmp10_AST);
        match(KEYWORD_WHILE);
        match(LPAREN);
        equality_expression();
        astFactory.addASTChild(currentAST, returnAST);
        match(RPAREN);
        compound_statement();
```

```
        astFactory.addASTChild(currentAST, returnAST);
        match(SEMI);
        loop_statement_AST = (AST)currentAST.root;
        returnAST = loop_statement_AST;
    }

    public final void null_statement() throws RecognitionException, Tok
enStreamException {

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

        match(SEMI);
        null_statement_AST = (AST)currentAST.root;
        returnAST = null_statement_AST;
    }

    public final void display_statement() throws RecognitionException,
TokenStreamException {

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

        AST tmp15_AST = null;
        tmp15_AST = astFactory.create(LT(1));
        astFactory.makeASTRoot(currentAST, tmp15_AST);
        match(KEYWORD_DISP);
        {
        switch ( LA(1)) {
        case ID:
        {
            identifier_expression();
            astFactory.addASTChild(currentAST, returnAST);
            break;
        }
        case DECLIT:
        case STRLIT:
        case HEXLIT:
        case OCTLIT:
        case BINLIT:
        case CHARLIT:
        {
            literal_expression();
            astFactory.addASTChild(currentAST, returnAST);
            break;
        }
        default:
        {
            throw new NoViableAltException(LT(1), getFilename());
        }
        }
        }
        match(SEMI);
        display_statement_AST = (AST)currentAST.root;
        returnAST = display_statement_AST;
```

```
    }

    public final void wait_statement() throws RecognitionException, Tok
enStreamException {

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

        AST tmp17_AST = null;
        tmp17_AST = astFactory.create(LT(1));
        astFactory.makeASTRoot(currentAST, tmp17_AST);
        match(KEYWORD_WAIT);
        AST tmp18_AST = null;
        tmp18_AST = astFactory.create(LT(1));
        astFactory.addASTChild(currentAST, tmp18_AST);
        match(DECLIT);
        match(SEMI);
        wait_statement_AST = (AST)currentAST.root;
        returnAST = wait_statement_AST;
    }

    public final void communications_statement() throws RecognitionExce
ption, TokenStreamException {

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

        switch ( LA(1)) {
        case KEYWORD_SEND:
        {
            send_statement();
            astFactory.addASTChild(currentAST, returnAST);
            communications_statement_AST = (AST)currentAST.root;
            break;
        }
        case KEYWORD_EXPECT:
        {
            expect_statement();
            astFactory.addASTChild(currentAST, returnAST);
            communications_statement_AST = (AST)currentAST.root;
            break;
        }
        case KEYWORD_INTERACT:
        {
            interact_statement();
            astFactory.addASTChild(currentAST, returnAST);
            communications_statement_AST = (AST)currentAST.root;
            break;
        }
        case KEYWORD_OPEN:
        {
            open_statement();
            astFactory.addASTChild(currentAST, returnAST);
            communications_statement_AST = (AST)currentAST.root;
            break;
```

```
        }
        case KEYWORD_CLOSE:
        {
            close_statement();
            astFactory.addASTChild(currentAST, returnAST);
            communications_statement_AST = (AST)currentAST.root;
            break;
        }
        default:
        {
            throw new NoViableAltException(LT(1), getFilename());
        }
        }
        returnAST = communications_statement_AST;
    }

    public final void compound_statement() throws RecognitionException,
TokenStreamException {

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

        match(LCURLY);
        {
        _loop656:
        do {
            switch ( LA(1)) {
            case KEYWORD_IF:
            {
                conditional_statement();
                astFactory.addASTChild(currentAST, returnAST);
                break;
            }
            case KEYWORD_WHILE:
            {
                loop_statement();
                astFactory.addASTChild(currentAST, returnAST);
                break;
            }
            case SEMI:
            {
                null_statement();
                astFactory.addASTChild(currentAST, returnAST);
                break;
            }
            case KEYWORD_DISP:
            {
                display_statement();
                astFactory.addASTChild(currentAST, returnAST);
                break;
            }
            case KEYWORD_WAIT:
            {
                wait_statement();
                astFactory.addASTChild(currentAST, returnAST);
                break;
```

```
            }
            case KEYWORD_SEND:
            case KEYWORD_EXPECT:
            case KEYWORD_INTERACT:
            case KEYWORD_OPEN:
            case KEYWORD_CLOSE:
            {
                communications_statement();
                astFactory.addASTChild(currentAST, returnAST);
                break;
            }
            default:
                if ((_tokenSet_1.member(LA(1))) && (LA(2)==SEMI||LA(2)=
=NOTEQUAL||LA(2)==EQUAL)) {
                    expression_statement();
                    astFactory.addASTChild(currentAST, returnAST);
                }
                else if ((LA(1)==ID) && (LA(2)==ASSIGNEQUAL)) {
                    assignment_statement();
                    astFactory.addASTChild(currentAST, returnAST);
                }
            else {
                break _loop656;
            }
            }
        } while (true);
        }
        match(RCURLY);
        compound_statement_AST = (AST)currentAST.root;
        compound_statement_AST = (AST)astFactory.make( (new ASTArray(2)
).add(astFactory.create(STMNT,"CTMNT")).add(compound_statement_AST));
        currentAST.root = compound_statement_AST;
        currentAST.child = compound_statement_AST!=null &&compound_stat
ement_AST.getFirstChild()!=null ?
            compound_statement_AST.getFirstChild() : compound_statement
_AST;
        currentAST.advanceChildToEnd();
        compound_statement_AST = (AST)currentAST.root;
        returnAST = compound_statement_AST;
    }

    public final void primary_expression() throws RecognitionException,
 TokenStreamException {

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

        switch ( LA(1)) {
        case DECLIT:
        case STRLIT:
        case HEXLIT:
        case OCTLIT:
        case BINLIT:
        case CHARLIT:
        {
            literal_expression();
```

```
            astFactory.addASTChild(currentAST, returnAST);
            primary_expression_AST = (AST)currentAST.root;
            break;
        }
        case PATLIT:
        {
            pattern_expression();
            astFactory.addASTChild(currentAST, returnAST);
            primary_expression_AST = (AST)currentAST.root;
            break;
        }
        default:
            if ((LA(1)==ID) && (LA(2)==SEMI)) {
                identifier_expression();
                astFactory.addASTChild(currentAST, returnAST);
                primary_expression_AST = (AST)currentAST.root;
            }
            else if ((LA(1)==ID) && (LA(2)==NOTEQUAL||LA(2)==EQUAL)) {
                equality_expression();
                astFactory.addASTChild(currentAST, returnAST);
                primary_expression_AST = (AST)currentAST.root;
            }
        else {
            throw new NoViableAltException(LT(1), getFilename());
        }
        }
        returnAST = primary_expression_AST;
    }

    public final void identifier_expression() throws RecognitionExcepti
on, TokenStreamException {

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

        AST tmp22_AST = null;
        tmp22_AST = astFactory.create(LT(1));
        astFactory.addASTChild(currentAST, tmp22_AST);
        match(ID);
        identifier_expression_AST = (AST)currentAST.root;
        returnAST = identifier_expression_AST;
    }

    public final void literal_expression() throws RecognitionException,
 TokenStreamException {

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

        {
        switch ( LA(1)) {
        case STRLIT:
        {
            AST tmp23_AST = null;
            tmp23_AST = astFactory.create(LT(1));
```

```
            astFactory.addASTChild(currentAST, tmp23_AST);
            match(STRLIT);
            break;
        }
        case CHARLIT:
        {
            AST tmp24_AST = null;
            tmp24_AST = astFactory.create(LT(1));
            astFactory.addASTChild(currentAST, tmp24_AST);
            match(CHARLIT);
            break;
        }
        case DECLIT:
        {
            AST tmp25_AST = null;
            tmp25_AST = astFactory.create(LT(1));
            astFactory.addASTChild(currentAST, tmp25_AST);
            match(DECLIT);
            break;
        }
        case HEXLIT:
        {
            AST tmp26_AST = null;
            tmp26_AST = astFactory.create(LT(1));
            astFactory.addASTChild(currentAST, tmp26_AST);
            match(HEXLIT);
            break;
        }
        case OCTLIT:
        {
            AST tmp27_AST = null;
            tmp27_AST = astFactory.create(LT(1));
            astFactory.addASTChild(currentAST, tmp27_AST);
            match(OCTLIT);
            break;
        }
        case BINLIT:
        {
            AST tmp28_AST = null;
            tmp28_AST = astFactory.create(LT(1));
            astFactory.addASTChild(currentAST, tmp28_AST);
            match(BINLIT);
            break;
        }
        default:
        {
            throw new NoViableAltException(LT(1), getFilename());
        }
        }
        }
        literal_expression_AST = (AST)currentAST.root;
        returnAST = literal_expression_AST;
    }

    public final void pattern_expression() throws RecognitionException,
TokenStreamException {
```

```
        returnAST = null;
        ASTPair currentAST = new ASTPair();
        AST pattern_expression_AST = null;

        AST tmp29_AST = null;
        tmp29_AST = astFactory.create(LT(1));
        astFactory.addASTChild(currentAST, tmp29_AST);
        match(PATLIT);
        pattern_expression_AST = (AST)currentAST.root;
        returnAST = pattern_expression_AST;
    }

    public final void equality_expression() throws RecognitionException
, TokenStreamException {

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

        identifier_expression();
        astFactory.addASTChild(currentAST, returnAST);
        {
        {
        switch ( LA(1)) {
        case NOTEQUAL:
        {
            AST tmp30_AST = null;
            tmp30_AST = astFactory.create(LT(1));
            astFactory.makeASTRoot(currentAST, tmp30_AST);
            match(NOTEQUAL);
            break;
        }
        case EQUAL:
        {
            AST tmp31_AST = null;
            tmp31_AST = astFactory.create(LT(1));
            astFactory.makeASTRoot(currentAST, tmp31_AST);
            match(EQUAL);
            break;
        }
        default:
        {
            throw new NoViableAltException(LT(1), getFilename());
        }
        }
        }
        literal_expression();
        astFactory.addASTChild(currentAST, returnAST);
        }
        equality_expression_AST = (AST)currentAST.root;
        returnAST = equality_expression_AST;
    }

    public final void send_statement() throws RecognitionException, Tok
enStreamException {

        returnAST = null;
```

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

        AST tmp32_AST = null;
        tmp32_AST = astFactory.create(LT(1));
        astFactory.makeASTRoot(currentAST, tmp32_AST);
        match(KEYWORD_SEND);
        {
        switch ( LA(1)) {
        case ID:
        {
            AST tmp33_AST = null;
            tmp33_AST = astFactory.create(LT(1));
            astFactory.addASTChild(currentAST, tmp33_AST);
            match(ID);
            break;
        }
        case STRLIT:
        {
            AST tmp34_AST = null;
            tmp34_AST = astFactory.create(LT(1));
            astFactory.addASTChild(currentAST, tmp34_AST);
            match(STRLIT);
            break;
        }
        case DECLIT:
        {
            AST tmp35_AST = null;
            tmp35_AST = astFactory.create(LT(1));
            astFactory.addASTChild(currentAST, tmp35_AST);
            match(DECLIT);
            break;
        }
        case HEXLIT:
        {
            {
            int _cnt671=0;
            _loop671:
            do {
                if ((LA(1)==HEXLIT)) {
                    AST tmp36_AST = null;
                    tmp36_AST = astFactory.create(LT(1));
                    astFactory.addASTChild(currentAST, tmp36_AST);
                    match(HEXLIT);
                }
                else {
                    if ( _cnt671>=1 ) { break _loop671; } else {throw n
ew NoViableAltException(LT(1), getFilename());}
                }

                _cnt671++;
            } while (true);
            }
            break;
        }
        case OCTLIT:
        {
```

```
            {
            int _cnt673=0;
            _loop673:
            do {
                if ((LA(1)==OCTLIT)) {
                    AST tmp37_AST = null;
                    tmp37_AST = astFactory.create(LT(1));
                    astFactory.addASTChild(currentAST, tmp37_AST);
                    match(OCTLIT);
                }
                else {
                    if ( _cnt673>=1 ) { break _loop673; } else {throw n
ew NoViableAltException(LT(1), getFilename());}
                }

                _cnt673++;
            } while (true);
            }
            break;
        }
        case BINLIT:
        {
            {
            int _cnt675=0;
            _loop675:
            do {
                if ((LA(1)==BINLIT)) {
                    AST tmp38_AST = null;
                    tmp38_AST = astFactory.create(LT(1));
                    astFactory.addASTChild(currentAST, tmp38_AST);
                    match(BINLIT);
                }
                else {
                    if ( _cnt675>=1 ) { break _loop675; } else {throw n
ew NoViableAltException(LT(1), getFilename());}
                }

                _cnt675++;
            } while (true);
            }
            break;
        }
        default:
        {
            throw new NoViableAltException(LT(1), getFilename());
        }
        }
        }
        match(SEMI);
        send_statement_AST = (AST)currentAST.root;
        returnAST = send_statement_AST;
    }

    public final void expect_statement() throws RecognitionException, T
okenStreamException {

        returnAST = null;
```

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

        AST tmp40_AST = null;
        tmp40_AST = astFactory.create(LT(1));
        astFactory.makeASTRoot(currentAST, tmp40_AST);
        match(KEYWORD_EXPECT);
        {
        switch ( LA(1)) {
        case DECLIT:
        case STRLIT:
        case HEXLIT:
        case OCTLIT:
        case BINLIT:
        case CHARLIT:
        case PATLIT:
        {
            valid_pairs_list_expect();
            astFactory.addASTChild(currentAST, returnAST);
            break;
        }
        case LCURLY:
        {
            {
            match(LCURLY);
            {
            int _cnt680=0;
            _loop680:
            do {
                if ((_tokenSet_2.member(LA(1)))) {
                    valid_pairs_list_expect();
                    astFactory.addASTChild(currentAST, returnAST);
                }
                else {
                    if ( _cnt680>=1 ) { break _loop680; } else {throw n
ew NoViableAltException(LT(1), getFilename());}
                }

                _cnt680++;
            } while (true);
            }
            match(RCURLY);
            }
            break;
        }
        default:
        {
            throw new NoViableAltException(LT(1), getFilename());
        }
        }
        }
        match(SEMI);
        expect_statement_AST = (AST)currentAST.root;
        returnAST = expect_statement_AST;
    }

    public final void interact_statement() throws RecognitionException,
```

```
 TokenStreamException {

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

        AST tmp44_AST = null;
        tmp44_AST = astFactory.create(LT(1));
        astFactory.makeASTRoot(currentAST, tmp44_AST);
        match(KEYWORD_INTERACT);
        {
        switch ( LA(1)) {
        case CHARLIT:
        {
            {
            valid_pairs_list_interact();
            astFactory.addASTChild(currentAST, returnAST);
            }
            break;
        }
        case LCURLY:
        {
            {
            match(LCURLY);
            {
            int _cnt700=0;
            _loop700:
            do {
                if ((LA(1)==CHARLIT)) {
                    valid_pairs_list_interact();
                    astFactory.addASTChild(currentAST, returnAST);
                }
                else {
                    if ( _cnt700>=1 ) { break _loop700; } else {throw n
ew NoViableAltException(LT(1), getFilename());}
                }

                _cnt700++;
            } while (true);
            }
            match(RCURLY);
            }
            break;
        }
        default:
        {
            throw new NoViableAltException(LT(1), getFilename());
        }
        }
        }
        match(SEMI);
        interact_statement_AST = (AST)currentAST.root;
        returnAST = interact_statement_AST;
    }

    public final void open_statement() throws RecognitionException, Tok
enStreamException {
```

```
        returnAST = null;
        ASTPair currentAST = new ASTPair();
        AST open_statement_AST = null;

        AST tmp48_AST = null;
        tmp48_AST = astFactory.create(LT(1));
        astFactory.makeASTRoot(currentAST, tmp48_AST);
        match(KEYWORD_OPEN);
        AST tmp49_AST = null;
        tmp49_AST = astFactory.create(LT(1));
        astFactory.addASTChild(currentAST, tmp49_AST);
        match(STRLIT);
        AST tmp50_AST = null;
        tmp50_AST = astFactory.create(LT(1));
        astFactory.addASTChild(currentAST, tmp50_AST);
        match(STRLIT);
        AST tmp51_AST = null;
        tmp51_AST = astFactory.create(LT(1));
        astFactory.addASTChild(currentAST, tmp51_AST);
        match(DECLIT);
        AST tmp52_AST = null;
        tmp52_AST = astFactory.create(LT(1));
        astFactory.addASTChild(currentAST, tmp52_AST);
        match(DECLIT);
        match(SEMI);
        open_statement_AST = (AST)currentAST.root;
        returnAST = open_statement_AST;
    }

    public final void close_statement() throws RecognitionException, To
kenStreamException {

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

        AST tmp54_AST = null;
        tmp54_AST = astFactory.create(LT(1));
        astFactory.makeASTRoot(currentAST, tmp54_AST);
        match(KEYWORD_CLOSE);
        match(SEMI);
        close_statement_AST = (AST)currentAST.root;
        returnAST = close_statement_AST;
    }

    public final void valid_pairs_list_expect() throws RecognitionExcep
tion, TokenStreamException {

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

        {
        switch ( LA(1)) {
        case DECLIT:
        {
```

```
            AST tmp56_AST = null;
            tmp56_AST = astFactory.create(LT(1));
            astFactory.addASTChild(currentAST, tmp56_AST);
            match(DECLIT);
            break;
        }
        case STRLIT:
        {
            AST tmp57_AST = null;
            tmp57_AST = astFactory.create(LT(1));
            astFactory.addASTChild(currentAST, tmp57_AST);
            match(STRLIT);
            break;
        }
        case CHARLIT:
        {
            AST tmp58_AST = null;
            tmp58_AST = astFactory.create(LT(1));
            astFactory.addASTChild(currentAST, tmp58_AST);
            match(CHARLIT);
            break;
        }
        case PATLIT:
        {
            AST tmp59_AST = null;
            tmp59_AST = astFactory.create(LT(1));
            astFactory.addASTChild(currentAST, tmp59_AST);
            match(PATLIT);
            break;
        }
        case HEXLIT:
        {
            {
            int _cnt684=0;
            _loop684:
            do {
                if ((LA(1)==HEXLIT)) {
                    AST tmp60_AST = null;
                    tmp60_AST = astFactory.create(LT(1));
                    astFactory.addASTChild(currentAST, tmp60_AST);
                    match(HEXLIT);
                }
                else {
                    if ( _cnt684>=1 ) { break _loop684; } else {throw n
ew NoViableAltException(LT(1), getFilename());}
                }

                _cnt684++;
            } while (true);
            }
            break;
        }
        case OCTLIT:
        {
            {
            int _cnt686=0;
            _loop686:
```

```
            do {
                if ((LA(1)==OCTLIT)) {
                    AST tmp61_AST = null;
                    tmp61_AST = astFactory.create(LT(1));
                    astFactory.addASTChild(currentAST, tmp61_AST);
                    match(OCTLIT);
                }
                else {
                    if ( _cnt686>=1 ) { break _loop686; } else {throw n
ew NoViableAltException(LT(1), getFilename());}
                }

                _cnt686++;
            } while (true);
            }
            break;
        }
        case BINLIT:
        {
            {
            int _cnt688=0;
            _loop688:
            do {
                if ((LA(1)==BINLIT)) {
                    AST tmp62_AST = null;
                    tmp62_AST = astFactory.create(LT(1));
                    astFactory.addASTChild(currentAST, tmp62_AST);
                    match(BINLIT);
                }
                else {
                    if ( _cnt688>=1 ) { break _loop688; } else {throw n
ew NoViableAltException(LT(1), getFilename());}
                }

                _cnt688++;
            } while (true);
            }
            break;
        }
        default:
        {
            throw new NoViableAltException(LT(1), getFilename());
        }
        }
        }
        valid_stmt_list();
        astFactory.addASTChild(currentAST, returnAST);
        valid_pairs_list_expect_AST = (AST)currentAST.root;

                valid_pairs_list_expect_AST = (AST)astFactory.make( (ne
w ASTArray(2)).add(astFactory.create(VPAIREX,"VPAIREX")).add(valid_pair
s_list_expect_AST));

        currentAST.root = valid_pairs_list_expect_AST;
        currentAST.child = valid_pairs_list_expect_AST!=null &&valid_pa
irs_list_expect_AST.getFirstChild()!=null ?
            valid_pairs_list_expect_AST.getFirstChild() : valid_pairs_l
```

```
ist_expect_AST;
        currentAST.advanceChildToEnd();
        valid_pairs_list_expect_AST = (AST)currentAST.root;
        returnAST = valid_pairs_list_expect_AST;
    }

    public final void valid_stmt_list() throws RecognitionException, To
kenStreamException {

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

        match(LCURLY);
        {
        int _cnt691=0;
        _loop691:
        do {
            switch ( LA(1)) {
            case KEYWORD_SEND:
            {
                send_statement();
                astFactory.addASTChild(currentAST, returnAST);
                break;
            }
            case KEYWORD_INTERACT:
            {
                interact_statement();
                astFactory.addASTChild(currentAST, returnAST);
                break;
            }
            case KEYWORD_WAIT:
            {
                wait_statement();
                astFactory.addASTChild(currentAST, returnAST);
                break;
            }
            case KEYWORD_WHILE:
            {
                loop_statement();
                astFactory.addASTChild(currentAST, returnAST);
                break;
            }
            case KEYWORD_DISP:
            {
                display_statement();
                astFactory.addASTChild(currentAST, returnAST);
                break;
            }
            case ID:
            {
                assignment_statement();
                astFactory.addASTChild(currentAST, returnAST);
                break;
            }
            case KEYWORD_IF:
            {
```

```
                conditional_statement();
                astFactory.addASTChild(currentAST, returnAST);
                break;
            }
            default:
            {
                if ( _cnt691>=1 ) { break _loop691; } else {throw new N
oViableAltException(LT(1), getFilename());}
            }
            }
            _cnt691++;
        } while (true);
        }
        match(RCURLY);
        valid_stmt_list_AST = (AST)currentAST.root;
        valid_stmt_list_AST = (AST)astFactory.make( (new ASTArray(2)).a
dd(astFactory.create(STMNT,"VSTMNT")).add(valid_stmt_list_AST));
        currentAST.root = valid_stmt_list_AST;
        currentAST.child = valid_stmt_list_AST!=null &&valid_stmt_list_
AST.getFirstChild()!=null ?
            valid_stmt_list_AST.getFirstChild() : valid_stmt_list_AST;
        currentAST.advanceChildToEnd();
        valid_stmt_list_AST = (AST)currentAST.root;
        returnAST = valid_stmt_list_AST;
    }

    public final void valid_stmt_list_interact() throws RecognitionExce
ption, TokenStreamException {

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

        match(LCURLY);
        {
        int _cnt694=0;
        _loop694:
        do {
            switch ( LA(1)) {
            case KEYWORD_SEND:
            {
                send_statement();
                astFactory.addASTChild(currentAST, returnAST);
                break;
            }
            case KEYWORD_WAIT:
            {
                wait_statement();
                astFactory.addASTChild(currentAST, returnAST);
                break;
            }
            case KEYWORD_WHILE:
            {
                loop_statement();
                astFactory.addASTChild(currentAST, returnAST);
                break;
            }
```

```
            case KEYWORD_DISP:
            {
                display_statement();
                astFactory.addASTChild(currentAST, returnAST);
                break;
            }
            case ID:
            {
                assignment_statement();
                astFactory.addASTChild(currentAST, returnAST);
                break;
            }
            case KEYWORD_IF:
            {
                conditional_statement();
                astFactory.addASTChild(currentAST, returnAST);
                break;
            }
            default:
            {
                if ( _cnt694>=1 ) { break _loop694; } else {throw new N
oViableAltException(LT(1), getFilename());}
            }
            }
            _cnt694++;
        } while (true);
        }
        match(RCURLY);
        valid_stmt_list_interact_AST = (AST)currentAST.root;
        valid_stmt_list_interact_AST = (AST)astFactory.make( (new ASTAr
ray(2)).add(astFactory.create(STMNT,"VSTMNT")).add(valid_stmt_list_inte
ract_AST));
        currentAST.root = valid_stmt_list_interact_AST;
        currentAST.child = valid_stmt_list_interact_AST!=null &&valid_s
tmt_list_interact_AST.getFirstChild()!=null ?
            valid_stmt_list_interact_AST.getFirstChild() : valid_stmt_l
ist_interact_AST;
        currentAST.advanceChildToEnd();
        valid_stmt_list_interact_AST = (AST)currentAST.root;
        returnAST = valid_stmt_list_interact_AST;
    }

    public final void valid_pairs_list_interact() throws RecognitionExc
eption, TokenStreamException {

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

        AST tmp67_AST = null;
        tmp67_AST = astFactory.create(LT(1));
        astFactory.addASTChild(currentAST, tmp67_AST);
        match(CHARLIT);
        valid_stmt_list_interact();
        astFactory.addASTChild(currentAST, returnAST);
        valid_pairs_list_interact_AST = (AST)currentAST.root;
```

```
                valid_pairs_list_interact_AST = (AST)astFactory.make( (
new ASTArray(2)).add(astFactory.create(VPAIRIN,"VPAIRIN")).add(valid_pa
irs_list_interact_AST));

        currentAST.root = valid_pairs_list_interact_AST;
        currentAST.child = valid_pairs_list_interact_AST!=null &&valid_
pairs_list_interact_AST.getFirstChild()!=null ?
            valid_pairs_list_interact_AST.getFirstChild() : valid_pairs
_list_interact_AST;
        currentAST.advanceChildToEnd();
        valid_pairs_list_interact_AST = (AST)currentAST.root;
        returnAST = valid_pairs_list_interact_AST;
    }


    public static final String[] _tokenNames = {
        "<0>",
        "EOF",
        "<2>",
        "NULL_TREE_LOOKAHEAD",
        "STMNT",
        "VPAIREX",
        "VPAIRIN",
        "LCURLY",
        "RCURLY",
        "SEMI",
        "ASSIGNEQUAL",
        "\"if\"",
        "LPAREN",
        "RPAREN",
        "\"else\"",
        "\"while\"",
        "\"disp\"",
        "\"wait\"",
        "DECLIT",
        "\"send\"",
        "ID",
        "STRLIT",
        "HEXLIT",
        "OCTLIT",
        "BINLIT",
        "\"expect\"",
        "CHARLIT",
        "PATLIT",
        "\"interact\"",
        "\"open\"",
        "\"close\"",
        "NOTEQUAL",
        "EQUAL",
        "\"catch\"",
        "\"true\"",
        "\"false\"",
        "WS",
        "NL",
        "COMMENT",
        "ALPHA",
        "DIGIT"
```

```
    };

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

    private static final long[] mk_tokenSet_0() {
        long[] data = { 2147453440L, 0L};
        return data;
    }
    public static final BitSet _tokenSet_0 = new BitSet(mk_tokenSet_0()
);
    private static final long[] mk_tokenSet_1() {
        long[] data = { 234094592L, 0L};
        return data;
    }
    public static final BitSet _tokenSet_1 = new BitSet(mk_tokenSet_1()
);
    private static final long[] mk_tokenSet_2() {
        long[] data = { 233046016L, 0L};
        return data;
    }
    public static final BitSet _tokenSet_2 = new BitSet(mk_tokenSet_2()
);

    }
```

9.3.3 TSVocabTokenTypes.java

```
// $ANTLR : "TSGrammar.g" -> "TSLexer.java"$

// stuff that is placed at the top of all generated files
package com.tingtech.tingstim;

/*
 * TSGrammar.g: the TingStim lexer and parser in ANTLR grammar
 *
 * @author Alvin Ting - at2337@columbia.edu | ating410@yahoo.com
 *
 */

public interface TSVocabTokenTypes {
    int EOF = 1;
    int NULL_TREE_LOOKAHEAD = 3;
    int STMNT = 4;
    int VPAIREX = 5;
    int VPAIRIN = 6;
    int LCURLY = 7;
    int RCURLY = 8;
    int SEMI = 9;
    int ASSIGNEQUAL = 10;
    int KEYWORD_IF = 11;
    int LPAREN = 12;
```

```
    int RPAREN = 13;
    int KEYWORD_ELSE = 14;
    int KEYWORD_WHILE = 15;
    int KEYWORD_DISP = 16;
    int KEYWORD_WAIT = 17;
    int DECLIT = 18;
    int KEYWORD_SEND = 19;
    int ID = 20;
    int STRLIT = 21;
    int HEXLIT = 22;
    int OCTLIT = 23;
    int BINLIT = 24;
    int KEYWORD_EXPECT = 25;
    int CHARLIT = 26;
    int PATLIT = 27;
    int KEYWORD_INTERACT = 28;
    int KEYWORD_OPEN = 29;
    int KEYWORD_CLOSE = 30;
    int NOTEQUAL = 31;
    int EQUAL = 32;
    int KEYWORD_CATCH = 33;
    int KEYWORD_TRUE = 34;
    int KEYWORD_FALSE = 35;
    int WS = 36;
    int NL = 37;
    int COMMENT = 38;
    int ALPHA = 39;
    int DIGIT = 40;
}
```

9.3.4 TSWalker.java

```
// $ANTLR : "TSWalker.g" -> "TSWalker.java"$

// stuff that is placed at the top of all generated files
package com.tingtech.tingstim;

/*
 * TSWalker.g: the TingStim walker in ANTLR grammar
 *
 * @author Alvin Ting - at2337@columbia.edu | ating410@yahoo.com
 *
 */


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 TSWalker extends antlr.TreeParser       implements TSWalke
rTokenTypes
 {

    private TSInterpreter tsi = new TSInterpreter();

    public void configure(String arg) {
        tsi.configure(arg);
    }

    public void exitProgie() {
        tsi.exitMethod("exit");
    }

public TSWalker() {
    tokenNames = _tokenNames;
}

    public final  String  progie(AST _t) throws RecognitionException {
         String r ;

        AST progie_AST_in = (_t == ASTNULL) ? null : (AST)_t;
        AST stmt = null;
        AST dstmt = null;
        AST wastmt = null;
        AST sstmt = null;
        AST instmt = null;
        AST inleft = null;
        AST inright = null;
        AST wleft = null;
        AST wright = null;
        AST ileft = null;
        AST thenp = null;
        AST elsep = null;
        AST vstmt = null;
        AST exleft = null;
        AST exright = null;
        AST ostmt = null;
        AST astmt = null;
        AST estmt = null;
        AST nestmt = null;
        AST dnum = null;
        AST hnum = null;
        AST onum = null;
        AST bnum = null;
        AST str = null;
        AST chr = null;
        AST id = null;
        String a = ""; r="";

        try {      // for error handling
            if (_t==null) _t=ASTNULL;
```

```
            switch ( _t.getType()) {
            case STMNT:
            {
                AST __t265 = _t;
                AST tmp1_AST_in = (AST)_t;
                match(_t,STMNT);
                _t = _t.getFirstChild();
                {
                _loop267:
                do {
                    if (_t==null) _t=ASTNULL;
                    if (((_t.getType() >= STMNT && _t.getType() <= SYM_
RT_ERROR))) {
                        stmt = (AST)_t;
                        if ( _t==null ) throw new MismatchedTokenExcept
ion();
                        _t = _t.getNextSibling();
                        r=progie(stmt);
                    }
                    else {
                        break _loop267;
                    }

                } while (true);
                }
                _t = __t265;
                _t = _t.getNextSibling();
                break;
            }
            case KEYWORD_DISP:
            {
                AST __t268 = _t;
                AST tmp2_AST_in = (AST)_t;
                match(_t,KEYWORD_DISP);
                _t = _t.getFirstChild();
                dstmt = (AST)_t;
                if ( _t==null ) throw new MismatchedTokenException();
                _t = _t.getNextSibling();
                _t = __t268;
                _t = _t.getNextSibling();
                tsi.dispMethod("disp", dstmt);
                break;
            }
            case KEYWORD_WAIT:
            {
                AST __t269 = _t;
                AST tmp3_AST_in = (AST)_t;
                match(_t,KEYWORD_WAIT);
                _t = _t.getFirstChild();
                wastmt = (AST)_t;
                if ( _t==null ) throw new MismatchedTokenException();
                _t = _t.getNextSibling();
                _t = __t269;
                _t = _t.getNextSibling();
                tsi.waitMethod("wait", wastmt);
                break;
            }
```

```
                case KEYWORD_SEND:
                {
                    AST __t270 = _t;
                    AST tmp4_AST_in = (AST)_t;
                    match(_t,KEYWORD_SEND);
                    _t = _t.getFirstChild();
                    sstmt = (AST)_t;
                    if ( _t==null ) throw new MismatchedTokenException();
                    _t = _t.getNextSibling();
                    _t = __t270;
                    _t = _t.getNextSibling();
                    tsi.sendMethod("send", sstmt);
                    break;
                }
                case KEYWORD_INTERACT:
                {
                    AST __t271 = _t;
                    AST tmp5_AST_in = (AST)_t;
                    match(_t,KEYWORD_INTERACT);
                    _t = _t.getFirstChild();
                    tsi.interactMethodExprBegin("interact");
                    {
                    _loop273:
                    do {
                        if (_t==null) _t=ASTNULL;
                        if (((_t.getType() >= STMNT && _t.getType() <= SYM_
RT_ERROR))) {
                            instmt = (AST)_t;
                            if ( _t==null ) throw new MismatchedTokenExcept
ion();
                            _t = _t.getNextSibling();
                            r=progie(instmt);
                        }
                        else {
                            break _loop273;
                        }

                    } while (true);
                    }
                    tsi.interactMethodEnd("interact");
                    _t = __t271;
                    _t = _t.getNextSibling();
                    break;
                }
                case VPAIRIN:
                {
                    AST __t274 = _t;
                    AST tmp6_AST_in = (AST)_t;
                    match(_t,VPAIRIN);
                    _t = _t.getFirstChild();
                    inleft = _t==ASTNULL ? null : (AST)_t;
                    r=progie(_t);
                    _t = _retTree;
                    tsi.interactMethodExprEnd("interact", inleft);
                    {
                    _loop276:
                    do {
```

```
                    if (_t==null) _t=ASTNULL;
                    if (((_t.getType() >= STMNT && _t.getType() <= SYM_
RT_ERROR))) {
                        inright = (AST)_t;
                        if ( _t==null ) throw new MismatchedTokenExcept
ion();
                        _t = _t.getNextSibling();
                        r=progie(inright);
                    }
                    else {
                        break _loop276;
                    }

                } while (true);
                }
                tsi.interactMethodStmtEnd("interact");
                _t = __t274;
                _t = _t.getNextSibling();
                break;
            }
            case KEYWORD_WHILE:
            {
                AST __t277 = _t;
                AST tmp7_AST_in = (AST)_t;
                match(_t,KEYWORD_WHILE);
                _t = _t.getFirstChild();
                tsi.whileMethodExprBegin("while");
                wleft = _t==ASTNULL ? null : (AST)_t;
                r=progie(_t);
                _t = _retTree;
                tsi.whileMethodExprEnd("while");
                {
                _loop279:
                do {
                    if (_t==null) _t=ASTNULL;
                    if (((_t.getType() >= STMNT && _t.getType() <= SYM_
RT_ERROR))) {
                        wright = (AST)_t;
                        if ( _t==null ) throw new MismatchedTokenExcept
ion();
                        _t = _t.getNextSibling();
                        r=progie(wright);
                    }
                    else {
                        break _loop279;
                    }

                } while (true);
                }
                tsi.whileMethodEnd("while");
                _t = __t277;
                _t = _t.getNextSibling();
                break;
            }
            case KEYWORD_IF:
            {
                AST __t280 = _t;
```

```
AST tmp8_AST_in = (AST)_t;
match(_t,KEYWORD_IF);
_t = _t.getFirstChild();
tsi.ifMethodExprBegin("if");
ileft = _t==ASTNULL ? null : (AST)_t;
r=progie(_t);
_t = _retTree;
tsi.ifMethodExprEnd("if");
thenp = (AST)_t;
if ( _t==null ) throw new MismatchedTokenException();
_t = _t.getNextSibling();
{
if (_t==null) _t=ASTNULL;
switch ( _t.getType()) {
case STMNT:
case VPAIREX:
case VPAIRIN:
case LCURLY:
case RCURLY:
case SEMI:
case ASSIGNEQUAL:
case KEYWORD_IF:
case LPAREN:
case RPAREN:
case KEYWORD_ELSE:
case KEYWORD_WHILE:
case KEYWORD_DISP:
case KEYWORD_WAIT:
case DECLIT:
case KEYWORD_SEND:
case ID:
case STRLIT:
case HEXLIT:
case OCTLIT:
case BINLIT:
case KEYWORD_EXPECT:
case CHARLIT:
case PATLIT:
case KEYWORD_INTERACT:
case KEYWORD_OPEN:
case KEYWORD_CLOSE:
case NOTEQUAL:
case EQUAL:
case KEYWORD_CATCH:
case KEYWORD_TRUE:
case KEYWORD_FALSE:
case WS:
case NL:
case COMMENT:
case ALPHA:
case DIGIT:
case SYM_NOT_EQUAL:
case SYM_EQUAL:
case SYM_LT_ERROR:
case SYM_RT_ERROR:
{
    elsep = (AST)_t;
```

```
                    if ( _t==null ) throw new MismatchedTokenException(
);
                    _t = _t.getNextSibling();
                    break;
                }
                case 3:
                {
                    break;
                }
                default:
                {
                    throw new NoViableAltException(_t);
                }
                }
                }
                _t = __t280;
                _t = _t.getNextSibling();

                                r = progie( thenp );
                                if ( null != elsep ) {
                                    tsi.elseMethodBegin("else");

                                    r = progie( elsep );
                                    tsi.elseMethodEnd("else");

                                } else {
                                    tsi.ifMethodNoElse("if");

                                }

                    break;
            }
            case KEYWORD_EXPECT:
            {
                AST __t282 = _t;
                AST tmp9_AST_in = (AST)_t;
                match(_t,KEYWORD_EXPECT);
                _t = _t.getFirstChild();
                tsi.expectMethodExprBegin("expect");
                {
                _loop284:
                do {
                    if (_t==null) _t=ASTNULL;
                    if (((_t.getType() >= STMNT && _t.getType() <= SYM_
RT_ERROR))) {
                        vstmt = (AST)_t;
                        if ( _t==null ) throw new MismatchedTokenExcept
ion();
                        _t = _t.getNextSibling();
                        r=progie(vstmt);
                    }
                    else {
                        break _loop284;
                    }

                } while (true);
                }
```

```
                    _t = __t282;
                    _t = _t.getNextSibling();
                        tsi.expectMethodNoCatch("expect");

                                    tsi.expectMethodEnd("expect");

                    break;
                }
                case VPAIREX:
                {
                    AST __t285 = _t;
                    AST tmp10_AST_in = (AST)_t;
                    match(_t,VPAIREX);
                    _t = _t.getFirstChild();
                    exleft = _t==ASTNULL ? null : (AST)_t;
                    r=progie(_t);
                    _t = _retTree;
                    tsi.expectMethodExprEnd("expect", exleft);
                    {
                    _loop287:
                    do {
                        if (_t==null) _t=ASTNULL;
                        if (((_t.getType() >= STMNT && _t.getType() <= SYM_
RT_ERROR))) {
                            exright = (AST)_t;
                            if ( _t==null ) throw new MismatchedTokenExcept
ion();

                            _t = _t.getNextSibling();
                            r=progie(exright);
                        }
                        else {
                            break _loop287;
                        }

                    } while (true);
                    }
                    tsi.expectMethodStmtEnd("expect");
                    _t = __t285;
                    _t = _t.getNextSibling();
                    break;
                }
                case KEYWORD_OPEN:
                {
                    AST __t288 = _t;
                    AST tmp11_AST_in = (AST)_t;
                    match(_t,KEYWORD_OPEN);
                    _t = _t.getFirstChild();
                    ostmt = (AST)_t;
                    if ( _t==null ) throw new MismatchedTokenException();
                    _t = _t.getNextSibling();
                    _t = __t288;
                    _t = _t.getNextSibling();
                    tsi.openMethod("open", ostmt);
                    break;
                }
                case KEYWORD_CLOSE:
                {
```

```
    AST __t289 = _t;
    AST tmp12_AST_in = (AST)_t;
    match(_t,KEYWORD_CLOSE);
    _t = _t.getFirstChild();
    {
    if (_t==null) _t=ASTNULL;
    switch ( _t.getType()) {
    case STMNT:
    case VPAIREX:
    case VPAIRIN:
    case ASSIGNEQUAL:
    case KEYWORD_IF:
    case KEYWORD_WHILE:
    case KEYWORD_DISP:
    case KEYWORD_WAIT:
    case DECLIT:
    case KEYWORD_SEND:
    case ID:
    case STRLIT:
    case HEXLIT:
    case OCTLIT:
    case BINLIT:
    case KEYWORD_EXPECT:
    case CHARLIT:
    case KEYWORD_INTERACT:
    case KEYWORD_OPEN:
    case KEYWORD_CLOSE:
    case NOTEQUAL:
    case EQUAL:
    {
        a=progie(_t);
        _t = _retTree;
        break;
    }
    case 3:
    {
        break;
    }
    default:
    {
        throw new NoViableAltException(_t);
    }
    }
    }
    _t = __t289;
    _t = _t.getNextSibling();
    System.out.println("close");
    break;
}
case ASSIGNEQUAL:
{
    AST __t291 = _t;
    AST tmp13_AST_in = (AST)_t;
    match(_t,ASSIGNEQUAL);
    _t = _t.getFirstChild();
    astmt = (AST)_t;
    if ( _t==null ) throw new MismatchedTokenException();
```

```
        _t = _t.getNextSibling();
        _t = __t291;
        _t = _t.getNextSibling();
        tsi.assignMethod("assign", astmt);
        break;
    }
    case EQUAL:
    {
        AST __t292 = _t;
        AST tmp14_AST_in = (AST)_t;
        match(_t,EQUAL);
        _t = _t.getFirstChild();
        estmt = (AST)_t;
        if ( _t==null ) throw new MismatchedTokenException();
        _t = _t.getNextSibling();
        _t = __t292;
        _t = _t.getNextSibling();
        tsi.equalMethod("equal", estmt);
        break;
    }
    case NOTEQUAL:
    {
        AST __t293 = _t;
        AST tmp15_AST_in = (AST)_t;
        match(_t,NOTEQUAL);
        _t = _t.getFirstChild();
        nestmt = (AST)_t;
        if ( _t==null ) throw new MismatchedTokenException();
        _t = _t.getNextSibling();
        _t = __t293;
        _t = _t.getNextSibling();
        tsi.notEqualMethod("notequal", nestmt);
        break;
    }
    case DECLIT:
    {
        dnum = (AST)_t;
        match(_t,DECLIT);
        _t = _t.getNextSibling();
        System.out.println("num is " + dnum.getText() );
        break;
    }
    case HEXLIT:
    {
        hnum = (AST)_t;
        match(_t,HEXLIT);
        _t = _t.getNextSibling();
        System.out.println("num is " + hnum.getText() );
        break;
    }
    case OCTLIT:
    {
        onum = (AST)_t;
        match(_t,OCTLIT);
        _t = _t.getNextSibling();
        System.out.println("num is " + onum.getText() );
        break;
```

```
        }
        case BINLIT:
        {
            bnum = (AST)_t;
            match(_t,BINLIT);
            _t = _t.getNextSibling();
            System.out.println("num is " + bnum.getText() );
            break;
        }
        case STRLIT:
        {
            str = (AST)_t;
            match(_t,STRLIT);
            _t = _t.getNextSibling();
            System.out.println("str is " + str.getText() );
            break;
        }
        case CHARLIT:
        {
            chr = (AST)_t;
            match(_t,CHARLIT);
            _t = _t.getNextSibling();
            System.out.println("chr is " + chr.getText() );
            break;
        }
        case ID:
        {
            AST __t294 = _t;
            id = _t==ASTNULL ? null :(AST)_t;
            match(_t,ID);
            _t = _t.getFirstChild();
            System.out.println(" id is " +  id.getText() );
            _t = __t294;
            _t = _t.getNextSibling();
            break;
        }
        default:
        {
            throw new NoViableAltException(_t);
        }
        }
    }
    catch (RecognitionException ex) {
        reportError(ex);
        if (_t!=null) {_t = _t.getNextSibling();}
    }
    _retTree = _t;
    return r ;
}


public static final String[] _tokenNames = {
    "<0>",
    "EOF",
    "<2>",
    "NULL_TREE_LOOKAHEAD",
    "STMNT",
```

```
        "VPAIREX",
        "VPAIRIN",
        "LCURLY",
        "RCURLY",
        "SEMI",
        "ASSIGNEQUAL",
        "\"if\"",
        "LPAREN",
        "RPAREN",
        "\"else\"",
        "\"while\"",
        "\"disp\"",
        "\"wait\"",
        "DECLIT",
        "\"send\"",
        "ID",
        "STRLIT",
        "HEXLIT",
        "OCTLIT",
        "BINLIT",
        "\"expect\"",
        "CHARLIT",
        "PATLIT",
        "\"interact\"",
        "\"open\"",
        "\"close\"",
        "NOTEQUAL",
        "EQUAL",
        "\"catch\"",
        "\"true\"",
        "\"false\"",
        "WS",
        "NL",
        "COMMENT",
        "ALPHA",
        "DIGIT",
        "SYM_NOT_EQUAL",
        "SYM_EQUAL",
        "SYM_LT_ERROR",
        "SYM_RT_ERROR"
    };

    }
```

9.3.5 TSWalkerTokenTypes.java

```
// $ANTLR : "TSWalker.g" -> "TSWalker.java"$

// stuff that is placed at the top of all generated files
package com.tingtech.tingstim;

/*
```

```
 * TSWalker.g: the TingStim walker in ANTLR grammar
 *
 * @author Alvin Ting - at2337@columbia.edu | ating410@yahoo.com
 *
 */


public interface TSWalkerTokenTypes {
    int EOF = 1;
    int NULL_TREE_LOOKAHEAD = 3;
    int STMNT = 4;
    int VPAIREX = 5;
    int VPAIRIN = 6;
    int LCURLY = 7;
    int RCURLY = 8;
    int SEMI = 9;
    int ASSIGNEQUAL = 10;
    int KEYWORD_IF = 11;
    int LPAREN = 12;
    int RPAREN = 13;
    int KEYWORD_ELSE = 14;
    int KEYWORD_WHILE = 15;
    int KEYWORD_DISP = 16;
    int KEYWORD_WAIT = 17;
    int DECLIT = 18;
    int KEYWORD_SEND = 19;
    int ID = 20;
    int STRLIT = 21;
    int HEXLIT = 22;
    int OCTLIT = 23;
    int BINLIT = 24;
    int KEYWORD_EXPECT = 25;
    int CHARLIT = 26;
    int PATLIT = 27;
    int KEYWORD_INTERACT = 28;
    int KEYWORD_OPEN = 29;
    int KEYWORD_CLOSE = 30;
    int NOTEQUAL = 31;
    int EQUAL = 32;
    int KEYWORD_CATCH = 33;
    int KEYWORD_TRUE = 34;
    int KEYWORD_FALSE = 35;
    int WS = 36;
    int NL = 37;
    int COMMENT = 38;
    int ALPHA = 39;
    int DIGIT = 40;
    int SYM_NOT_EQUAL = 41;
    int SYM_EQUAL = 42;
    int SYM_LT_ERROR = 43;
    int SYM_RT_ERROR = 44;
}
```

9.4 Java Source Files

9.4.1 TSConnectorPort.java

```
package com.tingtech.tingstim;

/*
 * TSConnectorPort.java: Used by TSInterpreter and generated
 * source java file to establish and maintain communications.
 *
 * @author Alvin Ting - at2337@columbia.edu | ating410@yahoo.com
 *
 */

import java.io.*;
import java.net.*;

public class TSConnectorPort {

    // private Serial tsSerial      = null;
    // private PrintWriter tsOut     = null;
    // private BufferedReader tsIn   = null;
    private DatagramSocket tsSendSocket = null;
    private DatagramSocket tsRecvSocket = null;
    private InetAddress addrURL     = null;
    private int addrPort            = 21;
    private boolean isConnected     = true;

    public TSConnectorPort() {
    } // end of TSConnectorPort

    /**
     * Open interface port with specified settings
     * @param portType - "Ethernet" (only one implemented,
     *                           "RS232", etc
     * @param otherLoc - Device host address
     * @param sendPort - Port number to send data.
     * @param recvPort - Port number to receive data.
     */
    public void openPort(String portType,
            String otherLoc, int sendPort, int recvPort) {
        boolean theResult = true;

        // For future revisions - Handle different port types
//        if (portType.equals("Ethernet")) {
//        } // end portType check

        try {
            addrURL  = InetAddress.getByName(otherLoc);
            addrPort = sendPort;

            System.out.println("Establishing " + portType
                    + " Connection to: " + addrURL);
```

```
            tsSendSocket = new DatagramSocket();
            tsRecvSocket = new DatagramSocket(recvPort);
            //tsOut = new PrintWriter(tsSocket.getOutputStream(), true)
;
            //tsIn  = new BufferedReader(new InputStreamReader(
            //    tsSocket.getInputStream()));
        } catch (UnknownHostException e) {
            System.out.println("Unknown Host : " + otherLoc);
            theResult = false;
        } catch (IOException e) {
            System.out.println("Unable to establish connection to "
                    + otherLoc);
            theResult = false;
        }
        isConnected = theResult;
        //return theResult;
    } // end openPort

    /**
     * Close all socket connections.
     *
     */
    public void closePort() {
        boolean theResult = true;

        //try {
            //tsOut.close();
            //tsIn.close();
            tsSendSocket.close();
            tsRecvSocket.close();
        //} catch (IOException o) {
        //    System.out.println("Error closing port.");
        //}
        isConnected = theResult;
        //return theResult;
    } // end of closePort

    /**
     * Send data over the socket
     * @param data - Data to be sent out over the coket
     */
    public void sendData(String data) {
        if (isConnected) {
            //tsOut.print(data);
            try {
                byte[] rawData = data.getBytes();
                DatagramPacket p = new DatagramPacket(rawData,
                        rawData.length, addrURL, addrPort);
                tsSendSocket.send(p);
            } catch (IOException o) {
                System.out.println("Error closing port.");
            }
        } else {
            System.out.println("Socket connection not " +
                    "established. Cannot send data.");
        }
    } // end of sendData
```

```
    /**
     * Reads data from socket. Blocking call.
     * @return String - data read from socket
     */
    public String recvData() {
        String theResult = null;
        if (isConnected) {
            try {
                byte[] buffer = new byte[4096];
                DatagramPacket p = new DatagramPacket(buffer,
                        buffer.length);
                tsRecvSocket.receive(p);
                theResult = new String(buffer, 0, p.getLength());
                //while ((theResult = tsIn.readLine()) == null) {}
            } catch (IOException o) {
                System.out.println("Error reading from port.");
            }
        } else {
            System.out.println("Socket connection not " +
                    "established. Cannot receive data.");
        }
        return theResult;
    } // end of recvData


    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub

    }

} // end of TSConnectorPort
```

9.4.2 TSInterpreter.java

```
package com.tingtech.tingstim;

/*
 * TSInterpreter.java: Called by TSWalker to translate
 * scenario statements into java statements
 *
 * @author Alvin Ting - at2337@columbia.edu | ating410@yahoo.com
 *
 */

import java.io.*;
import antlr.CommonAST;
import antlr.collections.AST;
import antlr.SemanticException;;
```

```
public class TSInterpreter implements TSWalkerTokenTypes {

    /* Store variables */
    private TSSymbolTable dict;
    /* Java Src file */
    private FileWriter scnFile;
    /* Java Src filename for Scenario */
    private String scnFileName;
    /* String to write to file */
    private String programString;
    /* Beautify indentation for scenario file */
    private int currIndent;

    /* Debug flag */
    private boolean debugFlag    = true;
    /* Semantic flag */
    private boolean semanticFlag = true;
    /* Open statement flag */
    private boolean openCalled   = false;

    // Constructor
    public TSInterpreter() {
        // Initialize variables
        dict = new TSSymbolTable();
        programString = "// TingStim Scenario Code\n";
        currIndent = 0;
    } // end of TSInterpreter

    /**
     * Configure interpreter prior to use
     * @param arg - Filename to use to determine name of
     *                  generated java source file
     */
    public void configure(String arg) {

        if (null == arg) {
            scnFileName = new String("testcase00");
        } else {
            scnFileName = arg;
        }

        try {
            scnFile = new FileWriter(scnFileName +
                    ".java",false); // do not append
        } catch (IOException ioe) {
            System.out.println("Failed to create file");
        }

        writePS("//package com.tingtech.tingstim;"); // TODO: Remove th
is code
        writePS("import com.tingtech.tingstim.*;");
        writePS("import java.io.*;");
        writePS("public class " + scnFileName +
                " implements Runnable {");
        incrIndent();
        writePS("private TSSymbolTable symTable;");
        writePS("private Thread currThread;");
```

```
        writePS("private String scratchString;");
        writePS("private TSConnectorPort tsPort;");
        writePS(scnFileName +"() {"); // Begin Constructor
        incrIndent();
        writePS("symTable = new TSSymbolTable();");
        writePS("tsPort = new TSConnectorPort();");
        decrIndent();
        writePS("}\n"); // End Constructor
        writePS("public void start() {"); // Begin start()
        incrIndent();
        writePS("if (null == currThread) {");
        incrIndent();
        writePS("currThread = new Thread(this, \""
                + scnFileName + "\");");
        writePS("currThread.start();");
        decrIndent();
        writePS("}");
        decrIndent();
        writePS("}\n"); // End start()
        writePS("public void stop() {"); // Begin stop()
        incrIndent();
        writePS("currThread = null;");
        decrIndent();
        writePS("}\n"); // End stop()
        writePS("public void run() {"); // Begin run()
        incrIndent();
//        writePS("Thread myThread = this.currenThread;");
//        writePS("while (currThread == myThread) {");
//        incrIndent();
        // GOTO EXIT METHOD

    } // end of configure

    /*#####################################
     *# Send method
     *#####################################*/

    /**
     * Send statement
     * @param stmt - send keyword
     * @param tree - what to send
     */
    public void sendMethod(String stmt, AST tree) {

        AST kids = tree;
        String packedData = new String();
        boolean convertToInt = false;
        int radix = 16;

        while (null != kids) {
            //writeDBG("Kids = " + kids.getText());
            switch(kids.getType())
            {
            case HEXLIT:
                convertToInt = true;
                packedData += (kids.getText()).substring(2,
                        kids.getText().length());
```

```
                break;
            case OCTLIT:
                radix = 8;
                convertToInt = true;
                packedData += (kids.getText()).substring(2,
                        kids.getText().length());

                break;
            case BINLIT:
                radix = 2;
                convertToInt = true;
                packedData += (kids.getText()).substring(2,
                        kids.getText().length());

                break;
            default:
                packedData += kids.getText();

                break;
            }
            kids = kids.getNextSibling();
        }
        // TODO: Check for Integer.MAX_VALUE ;

        //writeDBG(packedData);

        if (openCalled) {
            if (convertToInt) {
                try {
                    writePS("tsPort.sendData(\"" + (Integer.parseInt(
                            packedData, radix)) + "\");");
                } catch (NumberFormatException ee){
                    // Semantic error
                    writeERR("Exceeded integer maximum value: "
                            + Integer.MAX_VALUE);
                }
            } else {
                writePS("tsPort.sendData(\"" + packedData + "\");");

            }
        } else {
            writeERR("Must use open command prior to send.");
        }

    } // end of sendMethod

    /*#####################################
     *# Display method
     *#####################################*/

    /**
     * Display statement
     * @param stmt - display keyword
     * @param tree - what to display
     */
    public void dispMethod(String stmt, AST tree) {
        final String PREDISP = new String("System.out.println(\"");
```

```
        final String POSDISP = new String("\");");
        String midDisp  = tree.getText();
        int treeType = tree.getType();

//        writeDBG(    "Text: "     + midDisp
//                   + "\tType: " + treeType
//                   + "\tKids: " + tree.getNumberOfChildren()
//                );

        // Identifier lookup
        if (ID == treeType)
        {
            TSVariableType tmpVar = dict.getID(midDisp);
            if (null != tmpVar)
            {
                midDisp  = tmpVar.getData();
                treeType = tmpVar.getType();
            }
        }

        switch (treeType)
        {
        case STRLIT:
        case CHARLIT:
            writePS(PREDISP + midDisp + POSDISP);
            break;
        case DECLIT:
            writePS(PREDISP + midDisp + " [dec]" + POSDISP);

            break;
        case HEXLIT:
            writePS(PREDISP + midDisp.substring(2, midDisp.length())
                    + " [hex]" + POSDISP);
            break;
        case OCTLIT:
            writePS(PREDISP + midDisp.substring(2, midDisp.length())
                    + " [oct]" + POSDISP);
            break;
        case BINLIT:
            writePS(PREDISP + midDisp.substring(2, midDisp.length())
                    + " [bin]" + POSDISP);
            break;
        default:
            writeERR("Unknown type to display.");
            break;
        }

    } // end of dispMethod

    /*####################################
     *# Interact methods
     *####################################*/

    public void interactMethodExprBegin(String stmt) {
        writePS("// BEGIN INTERACT");
        writePS("try {");
        incrIndent();
```

```
        writePS("boolean gotSpecified = false;");
        writePS("while (!gotSpecified) {");
        incrIndent();
        writePS("scratchString = (new BufferedReader(" +
                "new InputStreamReader(System.in)).readLine());");
        writePS("if (false) { /* do nothing */ }");
    } // end of ifMethodExprBegin

    public void interactMethodExprEnd(String stmt, AST tree) {
        writePS("else if ( scratchString.equals(" +
                "String.valueOf(\'" + tree.getText() + "\')) )");
        writePS("{");
        incrIndent();
    } // end of interactMethodExpr

    public void interactMethodStmtEnd(String stmt) {
        writePS("gotSpecified = true;");
        decrIndent();
        writePS("}");
    } // end of interactMethodStmtEnd

    public void interactMethodEnd(String stmt) {
        writePS("else {");
        incrIndent();
        writePS("System.out.println(\"Unexpected character \" + " +
                "scratchString);");
        decrIndent();
        writePS("}");
        decrIndent();
        writePS("} // end of while");
        decrIndent();
        writePS("} catch (IOException ieo) {}");
        writePS("// END OF INTERACT");
    } // end of interactMethodEnd

    /*#####################################
     *# Expect methods
     *#####################################*/

    public void expectMethodExprBegin(String stmt) {
        if (openCalled) {
            writePS("// BEGIN EXPECT");
    //        writePS("try {");
    //        incrIndent();
            //writePS("scratchString = (new BufferedReader(" +
            //        "new InputStreamReader(System.in)).readLine());")
;
            writePS("scratchString = tsPort.recvData();");
            writePS("if (false) { /* do nothing */ }");
        } else {
            writeERR("Must call open command prior to expect.");
        }
    } // end of expectMethodExprBegin

    public void expectMethodExprEnd(String stmt, AST tree) {
        if (openCalled) {
            writePS("else if ( scratchString.equals(" +
```

```
                    "String.valueOf(\"" + tree.getText() + "\")) )");
            writePS("{");
            incrIndent();
        }
    } // end of expectMethodExpr

    public void expectMethodStmtEnd(String stmt) {
        if (openCalled) {
            decrIndent();
            writePS("}");
        }
    } // end of expectMethodStmtEnd

    public void expectMethodEnd(String stmt) {
        if (openCalled) {
//        decrIndent();
//        writePS("} catch (IOException ieo) {");
//        incrIndent();
//        writePS("System.out.println(\"Got I/O Error.\");");
//        decrIndent();
//        writePS("}");
            writePS("// END OF EXPECT");
        }
    } // end of expectMethodEnd

    public void expectMethodNoCatch(String stmt) {
        if (openCalled) {
            writePS("else {");
            incrIndent();
            writePS("System.out.println(\"Unexpected data received.\");
");
            decrIndent();
            writePS("}");
        }
    } // end of expectMethodNoCatch

    /*#####################################
     *# Catch methods
     *#####################################*/

    public void catchMethodBegin(String stmt) {
        writePS("else {");
        incrIndent();
    } // end of catchMethodBegin

    public void catchMethodEnd(String stmt) {
        decrIndent();
        writePS("}");
    } // end of catchMethodEnd

    /*#####################################
     *# Assignment method
     *#####################################*/

    /**
     * Assignment statement
     * @param stmnt - name of statement type
```

```
     * @param tree - subtree for this statement
     */
    public void assignMethod(String stmt, AST tree) {
        int leftNodeType = tree.getType();
        String leftString = tree.getText();

//        writeDBG(    "Text: "     + tree.getText()
//                + "\tType: " + tree.getType()
//                + "\tKids: " + tree.getNumberOfChildren()
//            );

        if (ID == leftNodeType) {
            AST rightNode = tree.getNextSibling();
            String rightString = rightNode.getText();
            int rightType = rightNode.getType();

            if (ID == rightType) {
                // Resolve right-hand identifier
                if (dict.checkID(rightString)) {
                    TSVariableType tmpV = null;
                    tmpV = dict.getID(rightString);
                    rightString = tmpV.getData();
                    rightType   = tmpV.getType();
                } else {
                    // semantic error
                    writeERR("Undefined right identifier in assignment.
");
                }
            }

            // Walker code
            dict.putNode(leftString, rightString, rightType);

            // Generate code
            writePS("symTable.putNode(\"" + leftString + "\" , \"" +
                    rightString + "\" , " + rightType + ");");

        } else {
            writeERR("Left assignment operand must be an identifier.");
        }
    } // end of assignMethod

    /*#####################################
     *# Equal method
     *#####################################*/

    /**
     * Equality statement
     * @param stmnt - name of statement type
     * @param tree - subtree for this statement
     */
    public void equalMethod(String stmt, AST tree) {

        // Note to self...
        // ID == ID (not useful as of now)
        // ID == LITERAL (OK)
```

```
        int leftType = tree.getType();
        String leftString = tree.getText();

//        writeDBG(    "Text: "     + tree.getText()
//                + "\tType: " + tree.getType()
//                + "\tKids: " + tree.getNumberOfChildren()
//            );

        if (ID == leftType) {
            AST rightNode = tree.getNextSibling();
            String rightString = rightNode.getText();
            int rightType = rightNode.getType();

            if (ID != rightType) {
                if ( dict.checkID(leftString) ) {
                    writePS("symTable.compareSymbolToRawData(\"" + left
String + "\" , \""
                            + rightString + "\" , " + rightType + ")");
                } else {
                    // Semantic error
                    writeERR("Undefined identifier.");
                }
            } else {
                // Semantic error
                writeERR("Right equality operand cannot be an identifie
r.");
            }

        } else {
            // Syntactical error
            writeERR("Left equality operand must be an identifier.");
        }
    } // end of equalMethod

    /*#####################################
     *# Not Equal method
     *#####################################*/

    /**
     * Equality statement
     * @param stmnt - name of statement type
     * @param tree - subtree for this statement
     */
    public void notEqualMethod(String stmt, AST tree) {
        int leftType = tree.getType();
        String leftString = tree.getText();

        if (ID == leftType) {
            AST rightNode = tree.getNextSibling();
            String rightString = rightNode.getText();
            int rightType = rightNode.getType();

            if (ID != rightType) {
                if ( dict.checkID(leftString) ) {
                    writePS("!(symTable.compareSymbolToRawData(\"" + le
ftString + "\" , \""
                            + rightString + "\" , " + rightType + "))")
```

```
;
                } else {
                    // Semantic error
                    writeERR("Undefined identifier.");
                }
            } else {
                // Semantic error
                writeERR("Right equality operand cannot be an identifie
r.");
            }

        } else {
            // Syntactical error
            writeERR("Left equality operand must be an identifier.");
        }
    } // end of notEqualMethod

    /*#####################################
     *# While methods
     *#####################################*/

    public void whileMethodExprBegin(String stmt) {
        writePS("while (");
    } // end of whileMethodExprBegin

    public void whileMethodExprEnd(String stmt) {
        writePS(") {");
        incrIndent();
    } // end of whileMethodExprEnd

    public void whileMethodEnd(String stmt) {
        decrIndent();
        writePS("}");
    } // end of whileMethodEnd

    /*#####################################
     *# If methods
     *#####################################*/

    public void ifMethodExprBegin(String stmt) {
        writePS("if (");
    } // end of ifMethodExprBegin

    public void ifMethodExprEnd(String stmt) {
        writePS(") {");
        incrIndent();
    } // end of ifMethodExprEnd

    public void ifMethodEnd(String stmt) {
        decrIndent();
        writePS("}");
    } // end of ifMethodEnd

    public void ifMethodNoElse(String stmt) {
        decrIndent();
        writePS("}");
    } // end of ifMethodNoElse
```

```
    /*#####################################
     *# Else methods
     *#####################################*/

    public void elseMethodBegin(String stmt) {
        decrIndent();
        writePS("} else {");
        incrIndent();
    } // end of elseMethodBegin

    public void elseMethodEnd(String stmt) {
        decrIndent();
        writePS("}");
    } // end of elseMethodEnd

    /*#####################################
     *# Wait method
     *#####################################*/

    /**
     * Wait for designated number of milliseconds
     * @param stmnt - name of statement type
     * @param tree - subtree for this statement
     */
    public void waitMethod(String stmt, AST tree) {

        String treeText  = tree.getText();
        int treeType     = tree.getType();

        if (DECLIT == treeType) {
            writePS("try { Thread.sleep(" + treeText + "); }");
            writePS("catch (InterruptedException et) {} ");
        } else {
            writeERR("Only decimal literals allowed after wait");
        }
    } // end of waitMethod

    /*#####################################
     *# Exit method
     *#####################################*/

    /**
     * Exit scenario
     * @param stmnt - name of statement type
     * @param tree - subtree for this statement
     */
    public void exitMethod(String stmt) {
//        decrIndent();
//        writePS("}// end of while"); // End of while
        decrIndent();
        writePS("}// end of run"); // End run()
        writePS("");
        writePS("public static void main(String[] args) {");
        incrIndent();
        writePS(scnFileName + " tsThread = new " +
                scnFileName + "();");
```

```
        writePS("tsThread.start();");
        decrIndent();
        writePS("}"); // End main()
        decrIndent();
        writePS("}"); // End of SCNFile()
    } // end of exitMethod

    /*#####################################
     *# Open method
     *#####################################*/

    /**
     * Open communication port
     * @param stmnt - name of statement type
     * @param tree - subtree for this statement
     */
    public void openMethod(String stmt, AST tree) {

        if (!openCalled) {
            String portType = tree.getText();
            String portLoc  = null;
            int sendPortNum = 0;
            int recvPortNum = 0;

            AST tmpVar  = tree.getNextSibling();
            portLoc     = tmpVar.getText();
            tmpVar      = tmpVar.getNextSibling();
            sendPortNum = Integer.parseInt(tmpVar.getText());
            tmpVar      = tmpVar.getNextSibling();
            recvPortNum = Integer.parseInt(tmpVar.getText());

            writePS("tsPort.openPort(\"" + portType + "\", \""
                    + portLoc + "\", " + sendPortNum + " , "
                    + recvPortNum + ");");
            openCalled = true;
        } else {
            writeERR("Can only call open once. Ignoring this one.");
        }

    } // end of openMethod

    /*#####################################
     *# Print methods
     *#####################################*/

    /**
     * Write string to file
     * @param ps - String to write to out buffer
     */
    private void writePS(String ps) {

        String niceStr = new String();
        int counter = currIndent;
        while(0 != counter) {
            niceStr += "\t";
            counter--;
        }
```

```
        niceStr += ps;
        niceStr += "\n";

        programString += niceStr;

        try {
            scnFile.write(niceStr);
            scnFile.flush();
        } catch (IOException ioe) {
            writeERR("Cannot write scenario file");
        }
    } // end of writePS

    /**
     * Write error string.
     * @param es - Error string to write.
     */
    private void writeERR(String es) {
        semanticFlag = false;
        System.out.println("ERR-> " + es);
    } // end of writeERR

    /**
     * Write debug string.
     * @param ds - Debug string to write
     */
    private void writeDBG(String ds) {
        if (debugFlag) {
            System.out.println("DBG->" + ds);
        }
    } // end of writeDBG

    /**
     * Write entire scenario
     */
    private void writeSCN() {
        try {
            scnFile.write(programString);
        } catch (IOException ioe) {
            writeERR("Cannot write scenario file");
        }
    } // end of writeSCN

    /*#####################################
     *# Misc methods
     *#####################################*/

    /**
     * Tab beautification
     */
    private void incrIndent() {
        currIndent++;
    } // end of incrIndent

    /**
     * Tab beautification
     */
```

```
    private void decrIndent() {
        currIndent--;
        if (currIndent<0) {
            writeERR("Indent counter miscount: " + currIndent);
        }
    } // end of decrIdent

    /**
     * Generate a unique string.
     * @return - String for unique variable name usage
     */
    private String getUnique() {
        return Long.toString(System.nanoTime());
    } // end of getUnique

} // end of TSInterpreter
```

9.4.3 TSMain.java

```
package com.tingtech.tingstim;

/*
 * TSMain.java: Launcher application.
 *
 * @author Alvin Ting - at2337@columbia.edu | ating410@yahoo.com
 *
 */

import java.io.*;
import antlr.*;

public class TSMain {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub

        String fileName = null;
        if (args.length > 0) {
            fileName = args[0];
            //System.out.println("Input file: " + fileName);
        }

        try {
            InputStream input = ( null != fileName ) ?
                    (InputStream) new FileInputStream( fileName ) :
                        (InputStream) System.in;

            if (null == fileName) {
                fileName = "testcase01";
                System.out.println("Type valid statement(s) and press C
```

```
TRL-Z to run.");
            } else {
                // Remove ".scn"
                fileName = fileName.substring(0, fileName.length() -
 4);
            }

            TSLexer myLexer = new TSLexer(input);
            TSParser myParser = new TSParser(myLexer);
            TSWalker myWalker = new TSWalker();

            myParser.progie();

            CommonAST myAST = (CommonAST)myParser.getAST();
            // Print the resulting tree out in LISP notation
            System.out.println(myAST.toStringList());

            //antlr.debug.misc.ASTFrame frame = new antlr.debug.misc.AS
TFrame("the tree", myAST);
            //frame.setVisible(true);

            myWalker.configure(fileName);
            myWalker.progie(myAST);
            myWalker.exitProgie();

        } catch( IOException e ) {
            System.err.println( "ERR-> I/O: " + e );
        } catch( RecognitionException e ) {
            System.err.println( "ERR-> Recognition: " + e );
        } catch( TokenStreamException e ) {
            System.err.println( "ERR-> Token stream: " + e );
        } catch( Exception e ) {
            System.err.println( "ERR-> " + e );
        }

        //System.exit(0);
    }
} // end of TSMain
```

9.4.4 TSSymbolTable.java

```
package com.tingtech.tingstim;

/*
 * TSSymbolTable.java: Used by TSInterpreter and generated
 * java source file to determine value and validity of
 * identifiers.
 *
 * @author Alvin Ting - at2337@columbia.edu | ating410@yahoo.com
 *
 */

import java.util.Hashtable;
```

```
public class TSSymbolTable implements TSWalkerTokenTypes {

    /* Store variables */
    private Hashtable dict;

    /*#####################################
     *# Hash methods
     *#####################################*/

    // Constructor
    public TSSymbolTable() {
        dict = new Hashtable();
    } // end of TSSymbolTable

    /**
     * @return if Key is in Hash
     */
    public boolean checkID(String idStr) {
        return dict.containsKey(idStr);
    }

    /**
     * Retrieve from Hash
     * @param idStr - Key into Hash
     * @return TSVariableType Object if found
     */
    public TSVariableType getID(String idStr) {
        TSVariableType tempVar = null;

        if (checkID(idStr)) {
            tempVar = (TSVariableType)(dict.get(idStr));
        } else {
            //System.out.println("Undefined identifier. Please assign v
alue prior to use.");
        }

        return tempVar;
    } // end of getID

    /**
     * Store in Hash
     * @param idName - Identifier name/key.
     * @param entry  - Contains data and data type.
     */
    public void putID(String idName, TSVariableType entry) {
        dict.put(idName, entry);
    } // end of putID

    /**
     * Compare two identifiers
     * @param key1 - key to first identifier
     * @param key2 - key to second identifier
     * @return -1,0,1 standard definition for compareTo...
     */
    public int compareSymbols(String key1, String key2) {
```

```
        int returnVal = SYM_NOT_EQUAL;
        TSVariableType tsv1 = (TSVariableType)dict.get(key1);
        TSVariableType tsv2 = (TSVariableType)dict.get(key2);

        if (null == tsv1) {
            returnVal = SYM_LT_ERROR;
        } else if (null == tsv1) {
            returnVal = SYM_RT_ERROR;
        } else {
            if (tsv1.getData().equals(tsv2.getData()))
            {
                returnVal = SYM_EQUAL;
            }
        }

        return returnVal;

    } // end of compareSymbols

    /**
     * Store value in Hash
     * @param key - key to use
     * @param data - Data to store
     * @param type - Type of the stored data
     */
    public void putNode(String key, String data, int type) {
        TSVariableType tempVar = new TSVariableType(
                data, type);
        putID(key, tempVar);
    } // end of putNode

    /**
     * Lookup identifier values and compare with passed in values
     * @param leftKey - key to matching identifier
     * @param rightData - String
     * @param rightType - int
     * @return true if identifier values and passed in are same
     */
    public boolean compareSymbolToRawData(String leftKey,
            String rightData, int rightType) {
        boolean theResult = true;

        TSVariableType leftVar = getID(leftKey);
        TSVariableType rightVar = new TSVariableType(
                    rightData, rightType);

        theResult = leftVar.equals(rightVar);

        //System.out.println("Result of compare is: " + theResult);

        return theResult;
    } // end of compareSymbolToRawData

} // end of TSSymbolTable
```

9.4.5 TSVariableType.java

```
package com.tingtech.tingstim;

/*
 * TSVariableType.java: Used by TSInterpreter to handle literals
 * and identifiers.
 *
 * @author Alvin Ting - at2337@columbia.edu | ating410@yahoo.com
 *
 */

public class TSVariableType implements TSWalkerTokenTypes {

    private String myData;
    private int myType;

    // Constructors
    public TSVariableType() {
        myData = new String();
        myType = 0;
    } // end of TSVariableType

    public TSVariableType(String data, int type) {
        myData = data;
        myType = type;
    } // end of TSVariableType

    /**
     * String - literal or identifier
     * @return - Store data string
     */
    public String getData() {
        return myData;
    } // end of getData

    /**
     * Type - See TokenTypes interface file for parser/walker
     * @return - Type of stored data
     */
    public int getType() {
        return myType;
    } // end of getType

    /**
     * Compare current attribute and another's
     * @param other - the object to compare with
     * @return - true if this and other object are equivalent
     */
    public boolean equals(TSVariableType other) {
        boolean theResult = true;

        String leftStr = getData();
        switch (getType()) {
            case BINLIT:
                leftStr = convertToDecLit(leftStr,2);
```

```
            break;
            case OCTLIT:
                leftStr = convertToDecLit(leftStr,8);
            case HEXLIT:
                leftStr = convertToDecLit(leftStr,16);
                break;
            default:
                break;
        } // end of switch

        String rightStr = other.getData();
        switch (other.getType()) {
            case BINLIT:
                rightStr = convertToDecLit(rightStr,2);
            break;
            case OCTLIT:
                rightStr = convertToDecLit(rightStr,8);
                break;
            case HEXLIT:
                rightStr = convertToDecLit(rightStr,16);
                break;
            default:
                break;
        } // end of switch

        //System.out.println("left = " + leftStr + " right = " + rightS
tr);

        theResult = leftStr.equals(rightStr);

//        System.out.println("left: " + this.getData() + " " + this.get
Type());
//        System.out.println("righ: " + other.getData() + " " + other.g
etType());
//        theResult = theResult && this.getData().equals(other.getData(
));
//        theResult = theResult && (this.getType() == other.getType());

        return theResult;
    } // end of equals

    /**
     * Convert from other number based literal to base 10
     * @param text - Original string
     * @param radix - Base of original number string
     * @return Decimal equivalent string
     */
    private String convertToDecLit(String text, int radix) {
        String theResult = text.substring(2, text.length());

        theResult = Integer.toString(Integer.parseInt(theResult, radix)
);

        return theResult;
    } // end of converToDecLit

} // end of TSVariableType
```

9.5 ANT Build Script

9.5.1 build.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="TingStim" default="jar" basedir=".">

        <!--
        build.xml: Build the tingstim program

        @author Alvin Ting - at2337@columbia.edu | ating410@yahoo.com
        -->
    <property name="srcdir"  value="com/tingtech/tingstim"/>
    <property name="destdir" value="build"/>
    <property name="deploy"  value="deploy"/>
        <property name="antlrfile" value="C:\utilities\antlr-
2.7.6\antlr.jar" />

        <target name="init">
                <!--
                <delete includeemptydirs="true">
                        <fileset dir="${destdir}" includes="**/*"/>
                </delete>
                -->
                <!--
                <delete includeemptydirs="true">
                        <fileset dir="${deploy}" includes="**/*"/>
                </delete>
                -->
                <mkdir dir="${deploy}"/>
                <mkdir dir="${destdir}"/>
        </target>

        <target name="tsgrammar">
                <antlr target="${srcdir}/TSGrammar.g"
                        outputdirectory="${destdir}" />
                <antlr target="${srcdir}/TSWalker.g"
                        outputdirectory="${destdir}" />
        </target>

    <target name="compile" depends="init,tsgrammar">
            <javac srcdir="${srcdir}" destdir="${destdir}"
                classpath="${antlrfile}" />
    </target>

        <target name="jar" depends="compile">
                <copy file="${antlrfile}" todir="${deploy}"/>
                <jar destfile="${deploy}/tingstim.jar"
basedir="${destdir}" >
                    <!-- excludes="**/*.java" > -->
```

```
                <manifest>
                                <attribute name="Built-By" value="Alvin
Ting"/>
                <attribute name="Main-Class"
value="com.tingtech.tingstim.TSMain"/>
                <attribute name="Class-Path" value="antlr.jar"/>
                </manifest>
                </jar>
        </target>

</project>
```

9.6 Test Scenarios

9.6.1 Unit Tests

9.6.1.1 test_disp.scn

```
1  /*
2   * test_disp.scn: Unit test for display statement.
3   *
4   * @author Alvin Ting - at2337@columbia.edu | ating410@yahoo.com
5   *
6   * CMD> java -jar tingstim.jar test_disp.scn
7   * CMD> javac -classpath tingstim.jar test_disp.java
8   * CMD> java test_disp
9   */
10
11 disp "Hello, world";
12 disp 4563;
13 disp 0x11D3;
14 disp 0o10723;
15 disp 0b1000111010011;
16 disp 'g';
17
18 /*
19 disp 566 6666;  // Only one literal per display statement
20                          // ERR-> Recognition: line 14:10: expecting
SEMI, found '6666';
21 disp ;  // Missing argument
22                  // ERR-> Recognition: line 16:6: unexpected token: ;
23 */
```

9.6.1.2 test_disp.java

```
//package com.tingtech.tingstim;
import com.tingtech.tingstim.*;
import java.io.*;
```

```
public class test_disp implements Runnable {
    private TSSymbolTable symTable;
    private Thread currThread;
    private String scratchString;
    private TSConnectorPort tsPort;
    test_disp() {
        symTable = new TSSymbolTable();
        tsPort = new TSConnectorPort();
    }

    public void start() {
        if (null == currThread) {
            currThread = new Thread(this, "test_disp");
            currThread.start();
        }
    }

    public void stop() {
        currThread = null;
    }

    public void run() {
        System.out.println("Hello, world");
        System.out.println("4563 [dec]");
        System.out.println("11D3 [hex]");
        System.out.println("10723 [oct]");
        System.out.println("1000111010011 [bin]");
        System.out.println("g");
    }// end of run

    public static void main(String[] args) {
        test_disp tsThread = new test_disp();
        tsThread.start();
    }
}
```

9.6.1.3 test_assign.scn

```
1  /*
2   * test_assign.scn: Unit test for assignment statement.
3   *
4   * @author Alvin Ting - at2337@columbia.edu | ating410@yahoo.com
5   *
6   * CMD> java -jar tingstim.jar test_assign.scn
7   * CMD> javac -classpath tingstim.jar test_assign.java
8   * CMD> java test_assign
9   */
10
11 animal = "monkey";
12 disp animal;
13
14 numAnimals = 3;
15 disp numAnimals;
16
17 numAnimals = 0x0003;
```

```
18 disp numAnimals;
19
20 mammal = "goat";
21 animal = mammal;
22 disp animal;
23
24 //10 = animal;  // Left operand must be an identifier
25                          // ERR-> Recognition: line 24:1: unexpected
token: test
26 //animal = unknownVar;  // Variable not defined prior to usage
27                                  // ERR-> Undefined right identifier
in assignment.
28 //disp unknownVar;      // Variable not defined prior to usage
29                          // ERR-> Unknown type to display.
30
```

9.6.1.4 test_assign.java

```
//package com.tingtech.tingstim;
import com.tingtech.tingstim.*;
import java.io.*;
public class test_assign implements Runnable {
    private TSSymbolTable symTable;
    private Thread currThread;
    private String scratchString;
    private TSConnectorPort tsPort;
    test_assign() {
        symTable = new TSSymbolTable();
        tsPort = new TSConnectorPort();
    }

    public void start() {
        if (null == currThread) {
            currThread = new Thread(this, "test_assign");
            currThread.start();
        }
    }

    public void stop() {
        currThread = null;
    }

    public void run() {
        symTable.putNode("animal" , "monkey" , 21);
        System.out.println("monkey");
        symTable.putNode("numAnimals" , "3" , 18);
        System.out.println("3 [dec]");
        symTable.putNode("numAnimals" , "0x0003" , 22);
        System.out.println("0003 [hex]");
        symTable.putNode("mammal" , "goat" , 21);
        symTable.putNode("animal" , "goat" , 21);
        System.out.println("goat");
    }// end of run

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

```
        test_assign tsThread = new test_assign();
        tsThread.start();
    }
}
```

9.6.1.5 test_if.scn

```
/*
 * test_if.scn: Unit test for conditional statement.
 *
 * @author Alvin Ting - at2337@columbia.edu | ating410@yahoo.com
 *
 * CMD> java -jar tingstim.jar test_if.scn
 * CMD> javac -classpath tingstim.jar test_if.java
 * CMD> java test_if
 */

animal = "monkey";

if (animal == "horse") {
        disp "I wanted a pony!";
} else {
        disp "You can't have a pony!";
};

if (animal != "horse") {
        disp "Haha, you didn't get a pony...";
};

if (animal == "monkey") {
        disp "you got a monkey instead!";
};

numToes = 10;
if (numToes == 0xA) {
        disp "That is equal.";
} else {
        disp "That is NOT equal.";
};

if (numToes == 0o666) {
        disp "That is equal.";
} else {
        disp "That is NOT equal.";
};

if (numToes == "test") {
        disp "That is equal.";
} else {
        disp "That is NOT equal.";
};

if (numToes != 't') {
        disp "that's good";
```

```
48 };
49
50
51 /*
52 if (numToes == unknownVar) {                    // Variable not
defined prior to usage
53         disp "Something wrong here.";           // ERR->
Recognition: line 35:16:
54 };                                                        //
unexpected token: unknownVar
55 */
56
```

9.6.1.6 test_if.java

```
//package com.tingtech.tingstim;
import com.tingtech.tingstim.*;
import java.io.*;
public class test_if implements Runnable {
    private TSSymbolTable symTable;
    private Thread currThread;
    private String scratchString;
    private TSConnectorPort tsPort;
    test_if() {
        symTable = new TSSymbolTable();
        tsPort = new TSConnectorPort();
    }

    public void start() {
        if (null == currThread) {
            currThread = new Thread(this, "test_if");
            currThread.start();
        }
    }

    public void stop() {
        currThread = null;
    }

    public void run() {
        symTable.putNode("animal" , "monkey" , 21);
        if (
        symTable.compareSymbolToRawData("animal" , "horse" , 21)
        ) {
            System.out.println("I wanted a pony!");
        } else {
            System.out.println("You can't have a pony!");
        }
        if (
        !(symTable.compareSymbolToRawData("animal" , "horse" , 21))
        ) {
            System.out.println("Haha, you didn't get a pony...");
        }
        if (
```

```
        symTable.compareSymbolToRawData("animal" , "monkey" , 21)
        ) {
            System.out.println("you got a monkey instead!");
        }
        symTable.putNode("numToes" , "10" , 18);
        if (
        symTable.compareSymbolToRawData("numToes" , "0xA" , 22)
        ) {
            System.out.println("That is equal.");
        } else {
            System.out.println("That is NOT equal.");
        }
        if (
        symTable.compareSymbolToRawData("numToes" , "0o666" , 23)
        ) {
            System.out.println("That is equal.");
        } else {
            System.out.println("That is NOT equal.");
        }
        if (
        symTable.compareSymbolToRawData("numToes" , "test" , 21)
        ) {
            System.out.println("That is equal.");
        } else {
            System.out.println("That is NOT equal.");
        }
        if (
        !(symTable.compareSymbolToRawData("numToes" , "t" , 26))
        ) {
            System.out.println("that's good");
        }
    }// end of run

    public static void main(String[] args) {
        test_if tsThread = new test_if();
        tsThread.start();
    }
}
```

9.6.1.7 test_while.scn

```
1  /*
2   * test_while.scn: Unit test for loop statement.
3   *
4   * @author Alvin Ting - at2337@columbia.edu | ating410@yahoo.com
5   *
6   * CMD> java -jar tingstim.jar test_while.scn
7   * CMD> javac -classpath tingstim.jar test_while.java
8   * CMD> java test_while
9   */
10
11
12 expression = "nothing";
13 while  (expression != "something") {
```

```
14          wait 1000;
15          disp "waiting...";
16  };
17
18  /*
19  while () {                      // Missing equality expression
20          disp "nothings";        // ERR-> Recognition: line 18:8:
expecting ID, found ‘)’
21  };
22  */
23
24  /*
25  while (expression == unknownVar) {              // Variable not
defined prior to usage
26          disp "Something wrong here.";           // ERR->
Recognition: line 35:16:
27  };
28  */                                                  //
unexpected token: unknownVar
29
30
```

9.6.1.8 test_while.java

```
//package com.tingtech.tingstim;
import com.tingtech.tingstim.*;
import java.io.*;
public class test_while implements Runnable {
    private TSSymbolTable symTable;
    private Thread currThread;
    private String scratchString;
    private TSConnectorPort tsPort;
    test_while() {
        symTable = new TSSymbolTable();
        tsPort = new TSConnectorPort();
    }

    public void start() {
        if (null == currThread) {
            currThread = new Thread(this, "test_while");
            currThread.start();
        }
    }

    public void stop() {
        currThread = null;
    }

    public void run() {
        symTable.putNode("expression" , "nothing" , 21);
        while (
        !(symTable.compareSymbolToRawData("expression" , "something" ,
21))
        ) {
```

```
            try { Thread.sleep(1000); }
            catch (InterruptedException et) {}
            System.out.println("waiting...");
        }
    }// end of run

    public static void main(String[] args) {
        test_while tsThread = new test_while();
        tsThread.start();
    }
}
```

9.6.1.9 test_wait.scn

```
/*
 * test_wait.scn: Unit test for wait statement.
 *
 * @author Alvin Ting - at2337@columbia.edu | ating410@yahoo.com
 *
 * CMD> java -jar tingstim.jar test_wait.scn
 * CMD> javac -classpath tingstim.jar test_wait.java
 * CMD> java test_wait
 */

expression = "nothing";
while  (expression != "something") {
        wait 1000;
        disp "tick";
};

/*
wait ;  // Missing millisecond argument
                // ERR-> Recognition: line 11:6: expecting DECLIT,
found ';';
*/


```

9.6.1.10 test_wait.java

```
//package com.tingtech.tingstim;
import com.tingtech.tingstim.*;
import java.io.*;
public class test_wait implements Runnable {
    private TSSymbolTable symTable;
    private Thread currThread;
    private String scratchString;
    private TSConnectorPort tsPort;
    test_wait() {
        symTable = new TSSymbolTable();
        tsPort = new TSConnectorPort();
```

```
    }

    public void start() {
        if (null == currThread) {
            currThread = new Thread(this, "test_wait");
            currThread.start();
        }
    }

    public void stop() {
        currThread = null;
    }

    public void run() {
        symTable.putNode("expression" , "nothing" , 21);
        while (
        !(symTable.compareSymbolToRawData("expression" , "something" ,
21))
        ) {
            try { Thread.sleep(1000); }
            catch (InterruptedException et) {}
            System.out.println("tick");
        }
    }// end of run

    public static void main(String[] args) {
        test_wait tsThread = new test_wait();
        tsThread.start();
    }
}
```

9.6.1.11 test_interact.scn

```
1  /*
2   * test_wait.scn: Unit test for interact statement.
3   *
4   * @author Alvin Ting - at2337@columbia.edu | ating410@yahoo.com
5   *
6   * CMD> java -jar tingstim.jar test_interact.scn
7   * CMD> javac -classpath tingstim.jar test_interact.java
8   * CMD> java test_interact
9   */
10
11 disp "Press b<ENTER> to continue";
12 interact 'b' { disp "got b"; };
13
14 killThis = "dont";
15 disp "Press a<ENTER> or b<ENTER> or 1<ENTER>";
16 while ( killThis == "dont" )
17 {
18         interact {
19                 'a' { disp "I believe I got an a"; }
20                 'b' { disp "I believe I got an b"; }
```

```
21                  '1' {
22                          disp "Exiting...";
23                          wait 2000;
24                          killThis = "exit";
25                  }
26          };
27 };
28
29 /*
30 interact 'a'                            // Interacts do not nest
31          interact 'b' {  disp "oh oh";   };   // ERR-> Recognition:
line 30:9:
32 };                  //     unexpected token: interact
33
34 interact "abc" { disp "something"; };   // Expect only character
literals
35                     // ERR-> Recognition: line 30:10:
36                     //     unexpected token: abc
37 */
38
```

9.6.1.12 test_interact.java

```
//package com.tingtech.tingstim;
import com.tingtech.tingstim.*;
import java.io.*;
public class test_interact implements Runnable {
    private TSSymbolTable symTable;
    private Thread currThread;
    private String scratchString;
    private TSConnectorPort tsPort;
    test_interact() {
        symTable = new TSSymbolTable();
        tsPort = new TSConnectorPort();
    }

    public void start() {
        if (null == currThread) {
            currThread = new Thread(this, "test_interact");
            currThread.start();
        }
    }

    public void stop() {
        currThread = null;
    }

    public void run() {
        System.out.println("Press b<ENTER> to continue");
        // BEGIN INTERACT
        try {
            boolean gotSpecified = false;
            while (!gotSpecified) {
                scratchString = (new BufferedReader(new InputStreamRead
```

```
er(System.in)).readLine());
                if (false) { /* do nothing */ }
                else if ( scratchString.equals(String.valueOf('b')) )
                {
                    System.out.println("got b");
                    gotSpecified = true;
                }
                else {
                    System.out.println("Unexpected character " + scratc
hString);
                }
            } // end of while
        } catch (IOException ieo) {}
        // END OF INTERACT
        symTable.putNode("killThis" , "dont" , 21);
        System.out.println("Press a<ENTER> or b<ENTER> or 1<ENTER>");
        while (
        symTable.compareSymbolToRawData("killThis" , "dont" , 21)
        ) {
            // BEGIN INTERACT
            try {
                boolean gotSpecified = false;
                while (!gotSpecified) {
                    scratchString = (new BufferedReader(new InputStream
Reader(System.in)).readLine());
                    if (false) { /* do nothing */ }
                    else if ( scratchString.equals(String.valueOf('a'))
 )
                    {
                        System.out.println("I believe I got an a");
                        gotSpecified = true;
                    }
                    else if ( scratchString.equals(String.valueOf('b'))
 )
                    {
                        System.out.println("I believe I got an b");
                        gotSpecified = true;
                    }
                    else if ( scratchString.equals(String.valueOf('1'))
 )
                    {
                        System.out.println("Exiting...");
                        try { Thread.sleep(2000); }
                        catch (InterruptedException et) {}
                        symTable.putNode("killThis" , "exit" , 21);
                        gotSpecified = true;
                    }
                    else {
                        System.out.println("Unexpected character " + sc
ratchString);
                    }
                } // end of while
            } catch (IOException ieo) {}
            // END OF INTERACT
        }
    }// end of run
```

```
    public static void main(String[] args) {
        test_interact tsThread = new test_interact();
        tsThread.start();
    }
}
```

9.6.1.13 test_send.scn

```
/*
 * test_send.scn: Unit test for send statement.
 *
 * @author Alvin Ting - at2337@columbia.edu | ating410@yahoo.com
 *
 * CMD> java -jar tingstim.jar test_send.scn
 * CMD> javac -classpath tingstim.jar test_send.java
 * CMD> java test_send
 */

//send "something"      // Missing open command prior to send
                        // ERR-> Must use open command prior to send.

open "Ethernet" "localhost" 2000 3000;

send "test";
send 100023;
send 0x0932 0x3432;
send 0o0003 0o3343 0o0001;
send 0b0101010101010101;

foobar = "barfoo";
send foobar;

foobar = 0xFF03;
send foobar;

/*
send 0b1010 0o04214 0o34243;    // Literals must be of same type
                                        // ERR-> Recognition: line
11:13:
                                        //    expecting SEMI, found
'0o04214'

send 0xFFFF 0xFFFF 0xFFFF 0xFFFF;       // Send integer size out of
bounds
                                                // ERR-> Exceeded
integer maximum
                                                //    value:
2147483647
*/
```

9.6.1.14 test_send.java

```
//package com.tingtech.tingstim;
import com.tingtech.tingstim.*;
import java.io.*;
public class test_send implements Runnable {
    private TSSymbolTable symTable;
    private Thread currThread;
    private String scratchString;
    private TSConnectorPort tsPort;
    test_send() {
        symTable = new TSSymbolTable();
        tsPort = new TSConnectorPort();
    }

    public void start() {
        if (null == currThread) {
            currThread = new Thread(this, "test_send");
            currThread.start();
        }
    }

    public void stop() {
        currThread = null;
    }

    public void run() {
        tsPort.openPort("Ethernet", "localhost", 2000 , 3000);
        tsPort.sendData("test");
        tsPort.sendData("100023");
        tsPort.sendData("154285106");
        tsPort.sendData("57552897");
        tsPort.sendData("21845");
        symTable.putNode("foobar" , "barfoo" , 21);
        tsPort.sendData("foobar");
        symTable.putNode("foobar" , "0xFF03" , 22);
        tsPort.sendData("foobar");
    }// end of run

    public static void main(String[] args) {
        test_send tsThread = new test_send();
        tsThread.start();
    }
}
```

9.6.1.15 test_expect.scn

```
1 /*
2  * test_expect.scn: Unit test for expect statement.
3  *
4  * @author Alvin Ting - at2337@columbia.edu | ating410@yahoo.com
5  *
6  * CMD> java -jar tingstim.jar test_expect.scn
7  * CMD> javac -classpath tingstim.jar test_expect.java
8  * CMD> java test_expect
```

```
 */

//expect 100 { disp "100"; };   // Must use open command before
expect
               // ERR-> Must call open command prior to expect.

open "Ethernet" "localhost" 5000 9000;

expect "something" { disp "I got something"; };

expect {
        0xFEED {
                wait 2000;
                testVar = "test";
                disp "testing";
        }
        "ad" { disp "got a"; }
        100304 { disp "got a number"; }
};

/*
expect "cool" {             // Expects do not nest
        expect "beans" {        disp "oh oh";   };
// ERR-> Recognition: line 29:9:
};           //     unexpected token: expect


expect 0xddd {// Only two valid expect forms allowed, can't mix them
        0xfdf { disp "oh oh"; } // ERR-> Recognition: line 29:9:
unexpected token: 0xfdf
};

*/
```

9.6.1.16 test_expect.java

```
//package com.tingtech.tingstim;
import com.tingtech.tingstim.*;
import java.io.*;
public class test_expect implements Runnable {
    private TSSymbolTable symTable;
    private Thread currThread;
    private String scratchString;
    private TSConnectorPort tsPort;
    test_expect() {
        symTable = new TSSymbolTable();
        tsPort = new TSConnectorPort();
    }

    public void start() {
        if (null == currThread) {
            currThread = new Thread(this, "test_expect");
            currThread.start();
        }
```

```
    }

    public void stop() {
        currThread = null;
    }

    public void run() {
        tsPort.openPort("Ethernet", "localhost", 5000 , 9000);
        // BEGIN EXPECT
        scratchString = tsPort.recvData();
        if (false) { /* do nothing */ }
        else if ( scratchString.equals(String.valueOf("something")) )
        {
            System.out.println("I got something");
        }
        else {
            System.out.println("Unexpected data received.");
        }
        // END OF EXPECT
        // BEGIN EXPECT
        scratchString = tsPort.recvData();
        if (false) { /* do nothing */ }
        else if ( scratchString.equals(String.valueOf("0xFEED")) )
        {
            try { Thread.sleep(2000); }
            catch (InterruptedException et) {}
            symTable.putNode("testVar" , "test" , 21);
            System.out.println("testing");
        }
        else if ( scratchString.equals(String.valueOf("ad")) )
        {
            System.out.println("got a");
        }
        else if ( scratchString.equals(String.valueOf("100304")) )
        {
            System.out.println("got a number");
        }
        else {
            System.out.println("Unexpected data received.");
        }
        // END OF EXPECT
    }// end of run

    public static void main(String[] args) {
        test_expect tsThread = new test_expect();
        tsThread.start();
    }
}
```

9.6.1.17 test_open.scn

```
1 /*
2  * test_expect.scn: Unit test for open statement.
3  *
4  * @author Alvin Ting - at2337@columbia.edu | ating410@yahoo.com
```

```
 5  *
 6  * CMD> java -jar tingstim.jar test_open.scn
 7  * CMD> javac -classpath tingstim.jar test_open.java
 8  * CMD> java test_open
 9  */
10
11 open "Ethernet" "localhost" 12000 10020;
12
13 /*
14 open "Ethernet" "localhost" 12000;      // Missing parameter
15                                                  // ERR->
Recognition: line 13:34:
16                                                  //    expecting
DECLIT, found ';'
17
18
19 open 10 10 "Ethernet" "test"; // Parameters are incorrect
20                                          // ERR-> Recognition: line
13:6:
21                                          //     expecting STRLIT,
found '10'
22
23
24 open "Ethernet" "localhost" 12001 10021;        // Only one call to
open
25                                                          // ERR-> Can
only call open once.
26                                                          //
Ignoring this one.
27 */
```

9.6.1.18 test_open.java

```
//package com.tingtech.tingstim;
import com.tingtech.tingstim.*;
import java.io.*;
public class test_open implements Runnable {
    private TSSymbolTable symTable;
    private Thread currThread;
    private String scratchString;
    private TSConnectorPort tsPort;
    test_open() {
        symTable = new TSSymbolTable();
        tsPort = new TSConnectorPort();
    }

    public void start() {
        if (null == currThread) {
            currThread = new Thread(this, "test_open");
            currThread.start();
        }
    }

    public void stop() {
```

```
        currThread = null;
    }

    public void run() {
        tsPort.openPort("Ethernet", "localhost", 12000 , 10020);
    }// end of run

    public static void main(String[] args) {
        test_open tsThread = new test_open();
        tsThread.start();
    }
}
```

9.6.2 Integration Tests

9.6.2.1 test_comms_client.scn

```
/*
 * test_comms_client.scn: Unit test for communication statements.
 *
 * @author Alvin Ting - at2337@columbia.edu | ating410@yahoo.com
 *
 * CMD> java -jar tingstim.jar test_comms_client.scn
 * CMD> javac -classpath tingstim.jar test_comms_client.java
 * CMD> java test_comms_client
 */


disp "Initializing client...";
disp "Opening port...";
open "Ethernet" "localhost" 5000 6000;

send "RTS";
expect "CTS" {
        disp "Server is ready";
        send 0b1010 0b1111 0b1110 0b1001;
};

disp "I'm going to kill the server!";
wait 1000;
send 0xDEAD;

wait 2000;
disp "done!";


```

9.6.2.2 test_comms_client.java

```
//package com.tingtech.tingstim;
import com.tingtech.tingstim.*;
import java.io.*;
```

```
public class test_comms_client implements Runnable {
    private TSSymbolTable symTable;
    private Thread currThread;
    private String scratchString;
    private TSConnectorPort tsPort;
    test_comms_client() {
        symTable = new TSSymbolTable();
        tsPort = new TSConnectorPort();
    }

    public void start() {
        if (null == currThread) {
            currThread = new Thread(this, "test_comms_client");
            currThread.start();
        }
    }

    public void stop() {
        currThread = null;
    }

    public void run() {
        System.out.println("Initializing client...");
        System.out.println("Opening port...");
        tsPort.openPort("Ethernet", "localhost", 5000 , 6000);
        tsPort.sendData("RTS");
        // BEGIN EXPECT
        scratchString = tsPort.recvData();
        if (false) { /* do nothing */ }
        else if ( scratchString.equals(String.valueOf("CTS")) )
        {
            System.out.println("Server is ready");
            tsPort.sendData("45033");
        }
        else {
            System.out.println("Unexpected data received.");
        }
        // END OF EXPECT
        System.out.println("I'm going to kill the server!");
        try { Thread.sleep(1000); }
        catch (InterruptedException et) {}
        tsPort.sendData("57005");
        try { Thread.sleep(2000); }
        catch (InterruptedException et) {}
        System.out.println("done!");
    }// end of run

    public static void main(String[] args) {
        test_comms_client tsThread = new test_comms_client();
        tsThread.start();
    }
}
```

9.6.2.3 test_comms_server.scn

```
/*
 * test_comms_client.scn: Unit test for communication statements.
 *
 * @author Alvin Ting - at2337@columbia.edu | ating410@yahoo.com
 *
 * CMD> java -jar tingstim.jar test_comms_server.scn
 * CMD> javac -classpath tingstim.jar test_comms_server.java
 * CMD> java test_comms_server
 */


disp "Initializing server...";
disp "Opening port...";
open "Ethernet" "localhost" 6000 5000;

killMe = "dont";
if (killMe == "dont")
{
        expect {
                "RTS" {
                        disp "Client is ready";
                        wait 2000;
                        send "CTS";
                }
                0xDEAD {
                        disp "Client killed me!";
                        wait 2000;
                        killMe = "DEAD";
                }
        };
};

wait 2000;
disp "done!";
```

9.6.2.4 test_comms_server.java

```
//package com.tingtech.tingstim;
import com.tingtech.tingstim.*;
import java.io.*;
public class test_comms_server implements Runnable {
    private TSSymbolTable symTable;
    private Thread currThread;
    private String scratchString;
    private TSConnectorPort tsPort;
    test_comms_server() {
        symTable = new TSSymbolTable();
        tsPort = new TSConnectorPort();
    }

    public void start() {
        if (null == currThread) {
            currThread = new Thread(this, "test_comms_server");
            currThread.start();
```

```
        }
    }

    public void stop() {
        currThread = null;
    }

    public void run() {
        System.out.println("Initializing server...");
        System.out.println("Opening port...");
        tsPort.openPort("Ethernet", "localhost", 6000 , 5000);
        symTable.putNode("killMe" , "dont" , 21);
        if (
        symTable.compareSymbolToRawData("killMe" , "dont" , 21)
        ) {
            // BEGIN EXPECT
            scratchString = tsPort.recvData();
            if (false) { /* do nothing */ }
            else if ( scratchString.equals(String.valueOf("RTS")) )
            {
                System.out.println("Client is ready");
                try { Thread.sleep(2000); }
                catch (InterruptedException et) {}
                tsPort.sendData("CTS");
            }
            else if ( scratchString.equals(String.valueOf("0xDEAD")) )
            {
                System.out.println("Client killed me!");
                try { Thread.sleep(2000); }
                catch (InterruptedException et) {}
                symTable.putNode("killMe" , "DEAD" , 21);
            }
            else {
                System.out.println("Unexpected data received.");
            }
            // END OF EXPECT
        }
        try { Thread.sleep(2000); }
        catch (InterruptedException et) {}
        System.out.println("done!");
    }// end of run

    public static void main(String[] args) {
        test_comms_server tsThread = new test_comms_server();
        tsThread.start();
    }
}
```

TingStim

[Final Report version 1.0]

END OF REPORT

Have a great summer!