NOPEL Project

Daniel Faltyn dtf2110@columbia.edu COMS 4115 PL Spring 2006

Introduction

NOPL is a **N**etwork **O**riented **P**rogramming **L**anguage that makes functions available over a network while abstracting the entire network stack from the developer. Server side programmers write functions using basic language constructs such as primitives, operators, and control structures. The language inherently makes the functions available over a network.

The NOPL compiler generates a .java file that is a socket-based implementation of the function. When compiled and run in Java the program listens on a specified port and executes the defined function(s) when it receives a socket request. A side-effect of the compilation is the generation of a .java proxy that clients can use to invoke the remote function(s.)

NOPL is designed for distributed applications that require data in a point-to-point fashion; i.e. clients call functions whenever they need to retrieve or manipulate data. For example, an address book java program may search for a person given a first and last name. A developer can write such a function in NOPL, compile and run the generated server as a Java executable, and give the generated proxy to the interested client(s.) The client can then call the function as if it were available in the local process space.

Background

The designer wanted to develop a language that various groups at an investment bank could use to prototype distributed systems. At Goldman Sachs, the Product Data Quality Group (PDQ) maintains the assets traders buy and sell, trader-support maintains a list of trader positions, HR maintains information about the traders, and Compliance oversees all the groups.

Java client applications in each of the areas need to query some or all of the services to get detailed information or perform a calculations. The following figure shows conceptually what developers would do to accommodate the given groups.

Developers first write three servers in NOPL: Trade Details Service, Trader Details Service and Product Details Service. The NOPL compiler generates three java files labeled “Separate Servers” that when further compiled and run in java, are immediately available on the network.

Client developers can embed any of the needed proxies (indicated with the name ‘%Proxy’ in the diagram) into their proprietary client code.

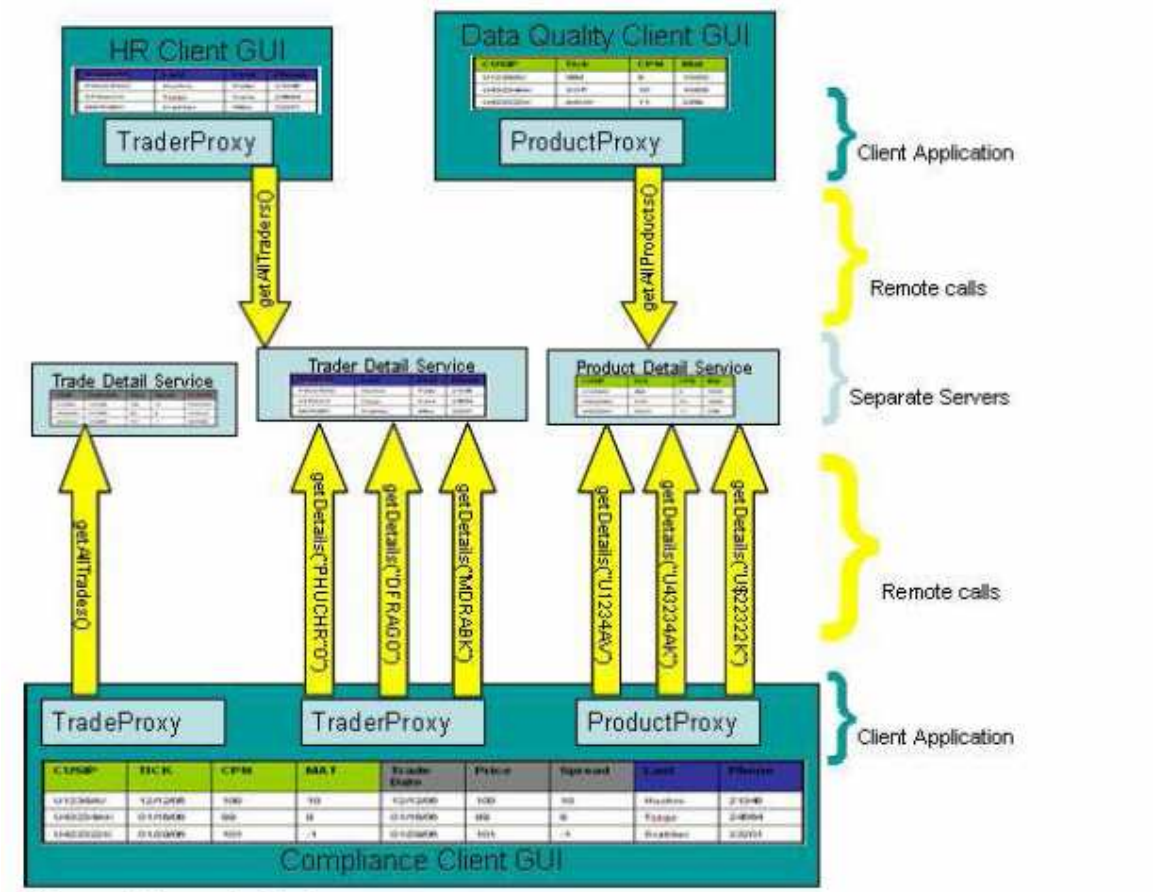


Figure 1 Example Environment

With this design implemented in NOPL, three separate groups can remotely access common functions via a common interface with the "Remote calls" abstracted by the NOPL language.

Goals

Network Abstraction

The primary goal of NOPL is to abstract the concept of the network from the programmer. Instead of writing socket based code and defining a protocol for client/server communication, developers define functions with the keyword *network* to make it available in a distributed environment. The following NOPL Hello World program compiled with the NOPL compiler generates a .java file that when compiled into java byte code and executed, listens on a port for requests to call this function.

```
server Hello
{
    network function helloWorld(string _name) returns string
    {
            return "Hello " + _name;
    }
}
```

Easy Access over a Network

To abstract the network from the client, the NOPL compiler generates a Java proxy that other programs use to call a function. The generated API is:

```
/**
* Autogenerated on: Thu May 04 21:25:31 GMT-05:00 2006
* Any changes you will be lost on re-compile.
*/
public class Hello
{
    protected String helloWorld(String _name )
    {
            return "Hello "+_name;
    }
    public static void main(String[] args) throws Exception
    {
            // ...
    }
}
```

And a client uses it as follows:

```
/**
* Autogenerated on: Thu May 04 21:25:31 GMT-05:00 2006
* Any changes you will be lost on re-compile.
*/
public class HelloClient
{
    public static void sethelloWorldConfig(String _ip, int _port)
    {
    }
    public static String helloWorld(String _name)
    {
    }
}
```

Easy to Learn

NOPL is a that implements a few basic data types (string, int, double, float, and date,) a few control structures (if/else and while,) a few collection constructs (structs and array,) and some basic mathematical operators.

Language Caveats

- Since the client program communicates with the server via a Java proxy, the client programmer must remember that any objects the proxy returns are passed by value rather than by reference. Specifically, changes made on the client side are NOT reflected on the server side.
- Without an object request broker (ORB) or naming service (JNDI) in place, client code must have *a priori* knowledge of the server location (IP address) and port.

Clients can implement extendable solutions by setting such parameters in the Java properties at run time. Future enhancements to the language can include usage of a naming service similar to Java RMI.

Example Code

This is code written in NOPL.

```
server TraderDetailsService
{
    struct Trader
    {
            string fName;
            string lName;
            string desk;
            real salary;
    };
    Trader headTrader;
    headTrader.lName     = "Huchro";
    headTrader.desk      = "IGC";
    headTrader.salary    = 25000000;

    Trader juniorTrader;
    juniorTrader.lName   = "Robinson";
    juniorTrader.desk    = "IGC";
    juniorTrader.salary  = 5000000;

    Trader traders[2];
    traders[0] = headTrader;
    traders[1] = juniorTrader;

    network function getAllTraders() returns Trader[]
    { return traders;}
    network function getDetails(int _traderId) returns Trader
    {
            Trader t;
            if (_traderId == 0){t = traders[0];}
            else {t = traders[1];}
            return t;
    }
}
```

This is the output of the NOPL compiler (the server.)

```
/**
* Autogenerated on: Thu May 04 22:19:04 GMT-05:00 2006
* Any changes you will be lost on re-compile.
*/
public class TraderDetailsService
{
```

```
    protected Trader[] getAllTraders( ){// ...}
    protected Trader getDetails(int _traderId ){ //...}
    public static void main(String[] args) throws Exception{ //...}
}
```

This is the output of the NOPL compiler (the client.)

```
/**
* Autogenerated on: Thu May 04 22:19:04 GMT-05:00 2006
* Any changes you will be lost on re-compile.
*/
public class TraderDetailsServiceClient
{
    public static void setgetAllTradersConfig(String _ip, int _port){// ...}
    public static Trader[] getAllTraders(){// ...}
    public static void setgetDetailsConfig(String _ip, int _port){// ...}
    public static Trader getDetails(int _traderId){// ...}
}
```

Language Tutorial

The steps to writing, compiling and running a NOPEL program are:

1. Write a NOPEL program and store it in a file
2. Compile the NOPEL file into a java client and java server
3. Compile the java server into a java class file
4. Write a driver for the client code that uses the generated client stub to call the server
5. Execute the server java class file
6. Execute the driver

Hello World Example

1. Create a file in some directory named hello.n that has the following contents

```
server Hello
{
    network function hello( string _name ) returns string
{
    // Log so we see something happening on the server
    LOG_INFO("Some is calling function hello with name: " + _name);

    return "Hello " + _name;

}
}
```

2. Compile the file into a java client and server by typing:
    a. java –jar NOPELCompiler.java ./ hello.n
    b. you should see 2 files get generated Hello.java and HelloClient.java
3. Compile the java server file (Hello.java) by typing: javac –cp . Hello.java
4. Run the server by typing: java –cp . Hello

a. You will be prompted to enter a port that the function should listen on.
b. Enter in a free port and the service should indicate it is running

5. Compile the client by typing: javac –cp . HelloClient.java
6. This client has no main so you will have to create a driver in java.
    a. Create a java file named driver.java
    b. Copy the following text to that file.

```
public class driver
{
    public static void main(String _args[])
    {

            HelloClient.sethelloConfig("127.0.0.1",123);
            System.out.println(HelloClient.hello("Daniel"));
    }
}
```

7. Compile this driver in the same directory as HelloClient.java by typing: javac –cp . driver.java
8. Run the client program by typing java –cp . driver
    a. The client uses/requires a SAX implementation such as crimson so make sure one is defined on your classpath. See an appendix for more details. (The CLIC machines all have an implementation on the classpath.)
9. You should see the server print out a message on the server console and the client print out a message on the client console.

Detailed Example

This example show a server that has three functions:

1. hello(person p) : takes a person and prints out a message with that persons first and last name and then returns that message string.
2. helloPeople(person p[]) : takes an array of people and prints out a message with all their names, then returns that message string
3. helloSame(person p[]) : takes an array of people, prints out the number of people in the array then returns a copy of that array to the caller

1. Create a file in some directory named hello.n that has the following contents

```
server Hello
{
    struct person { string fName; string lName;};
    network function hello( person p ) returns string
{
    // Log so we see something happening on the server
    LOG_INFO("Some is calling function hello with name: " + p.fName);

    return "Hello " + p.fName+ " " + p.lName;

}
    network function helloPeople( person p[] ) returns string
```

```
{
    // Log so we see something happening on the server
    LOG_INFO("Some is calling function helloPeople with: " + p.length+ " people");
    int i = 0;
    string omg = "Hello: ";
    while ( i < p.length )
    {
            omg = (omg + p[i].fName + " "+ p[i].lName+" ");
            i = i + 1;
    }

    return omg;
}
    network function helloSame( person p[] ) returns person[]
{
    // Log so we see something happening on the server
    LOG_INFO("Some is calling function helloSame with: " + p.length+ " people");
    return p;
}
}
```

2. Compile the file into a java client and server by typing:
    a. java –jar NOPELCompiler.java ./ hello.n
    b. you should see 2 files get generated Hello.java and HelloClient.java
3. Compile the java server file (Hello.java) by typing: javac –cp . Hello.java
4. Run the server by typing: java –cp . hello
    a. You will be prompted to enter a port that the function should listen on.
    b. Enter in a free port and the service should indicate it is running
5. Compile the client by typing: javac –cp . HelloClient.java
6. This client has no main so you will have to create a driver in java.
    a. Create a java file named driver.java
    b. Copy the following text to that file.

```
public class driver
{
    public static void main(String _args[])
{
            HelloClient.sethelloConfig("127.0.0.1", 1222);
            HelloClient.sethelloPeopleConfig("127.0.0.1", 1223);
            HelloClient.sethelloSameConfig("127.0.0.1", 1224);

            // Make the remote call
            HelloClient.person p = (new HelloClient()).new person();
            p.fName = "Stephen";
            p.lName = "Edwards";
            String s = HelloClient.hello(p);

            System.out.println(s);
```

```
            System.out.println("Now the helloPeopleConfig call....");

            HelloClient.person p2 = (new HelloClient()).new person();
            p2.fName = "Dan";
            p2.lName = "Faltyn";
            HelloClient.person array[] = {p,p2};
            s = HelloClient.helloPeople(array);

            System.out.println(s);

            System.out.println("Now the helloSame call....");

            HelloClient.person[] sArray = HelloClient.helloSame(array);
            System.out.println("Got back these people:");
            for (int i=0; i < sArray.length; ++i)
            {
                    System.out.println(sArray[i].fName +" "+ sArray[i].lName);
            }
    }
}
```

Language Reference Manual

This manual describes the NOPEL (Network Oriented Programming Language) lexical conventions and syntax notation. Though similar to C and Java, there are some subtle differences outlined in this document. After writing a program that is compliant with these rules, the first phase of the compilation process generates a sequence of tokens that are later translated into a Java server file and client file. Upon successful compilation with a Java compiler, a programmer will have at his disposal functions that are available over a network and means to call those functions.

```
// A word on notation:  Sections like this in grey are code snippets.
```

Lexical Clambakes (Conventions)

A NOPEL program consists of a single file that has one or more network functions (functions that are available over a network) and zero or more local functions that are visible to other functions (including network functions) in the same file. These functions and various global variables are all parsed into a stream of tokens as described in the following section.

Tokens

. The broad categories of tokens in NOPEL are comments, identifiers, keywords, numbers, strings, and dates. The NOPEL lexer ignores white space, new lines and tabs except where they separate tokens as described below. The NOPELLexer similarly

recognizes and ignores comment and various separators such as commas, and curly braces.

Comments

Comments in NOPEL come in the usual 2 flavors: those beginning with a double slash "//" that begins a comment and terminates at the following new line and comments that begin with a slash star "/*" and span on or more lines until a star slash "*/" is encountered.

```
// this is a single line comment in NOPEL. It terminates with a new line.
/* this is a multi-line comment in NOPEL. It can be on one line */
/* or it can span
   multiple lines */
```

Identifiers

An identifier consists of a sequence of letters, digits and the underscore character beginning with an underscore or a character. Case is significant so the identifier *myVariable* is distinct from *MyVariable*.

```
int g_globalVar = 10; // the identifier is g_globalVar and holds the integer val 10
```

Keywords

The NOPEL language is translated into Java so all Java keywords[1] are also reserved. The following keywords are explicitly defined in NOPEL and programmers may not use them other than for their intended usage.

function	network	date	true	False
if	else	while	break	Continue
return	returns	struct	void	LOG_FATAL
LOG_INFO	LOG_WARN	LOG_ERROR		

Numbers

The NOPEL lexer recognizes Integers as a sequence of one or more digits and Real numbers as one or more digits followed by a "." followed by one or more digits. The following are various examples of NOPEL numbers.

```
1  // is an integer
2222 // is an integer
2.3 // is a real
222.2222 // is a real
```

Strings

The NOPEL lexer recognizes strings as a sequence of characters surrounded by double quotes. Commonly escaped characters t, b, n, \, and the double quote are allowed as long as they are prefixed with a \.

```
“This is a string with an escaped backslash as the last character \\”
```

Dates

The NOPEL lexer recognizes dates as **ddMMMyyyy** where d and y are digits and M are alphabetic characters. The first 2 digits represent the day of the month, the 3 characters are the first 3 characters of any of the 12 months, and the final 4 digits are the year.

The Lexer does not differentiate the case for the month characters so 12FEB2006 and 12Feb2006 are interpreted the same. The Lexer does not accept dates that include invalid days of the month (e.g. 45Mar2006), bad months (e.g. 05ZOR2006), or abbreviated years (e.g. 23Mar06.)

```
02Mar2006 // this is a valid date.
```

Convenience Tokens

NOPEL recognizes these other common tokens for the sake of punctuation and clarity.

Type	Comment
{ }	Demarcates the beginning and end of a function or enumeration
[ ]	Demarcates the beginning an end of an array
;	Separates expressions
,	Separates parameters in functions

Syntax Notation

This section addresses the interpretation of the tokens listed in the previous section. NOPEL is a statically typed language so each object must have a declared type that is immutable for the duration of its existence. The second section deals with the usage of the types in a variety of expressions and the third section outlines the constructs required for logical programming.

```
// A word on notation:  Sections like this in light blue are syntax
```

Types

There are three classifications of types in NOPEL: primitives, structures, and arrays.

Primitive Types

The following table lists the NOPEL primitive types, the type it corresponds to in Java, and an example usage as a variable.

Type	Java conversion	Default value	Usage
boolean	Boolean	False	boolean isANiceDay = true;
int	Int	0	int age = 29;
real	Double	0.0	real weight = 180.22;
string	java.lang.String	“”	string name = “Daniel”;

date	java.util.Date	todays date[2]	date today = 18Feb2006;

Structures

A structure is a grouping of named primitive types into a single object that can be referenced as a unit. The declaration of a structure is achieved though the keyword struct, a left curly, a definition of one or more typed fields, and a right curly brace.

A NOPEL program must declare a structure before usage. Programs can reference fields in the structure with the usual "dot" notation of C.

```
struct
{
        ((boolean | int | real | string | date) identifier ";")*
}
```

The following example shows a declaration for a structure that contains information about a trader followed by an example usage.

```
// declaration of the structure type trader
struct trader
{
   string firstName;
   string lastName;
   int age;
   date startDate;
   string desk;
};
trader bob; // instance of type trader
bob.firstName = "Robert";     // assigning the value of the field firstName
int age = bob.age;            // retrieving the value of the field age.
```

Arrays

NOPEL provides the means to group primitive types and structures into typed arrays of a fixed size. Arrays have the property **length** that indicates the size of the array as an integer. Programs can retrieve values from an array by accessing the values indexed from 0 to length -1 as shown.

```
(boolean | int | real | string | date) identifier "[" int "]""";"
```

```
// create an array of size 10 (default all values initialized to 0)
int accountNumbers[10];
int arrayLen = accountNumbers.length; // returns the integer 10.
```

[2] The date returned is the date as of the running of the program.

Expressions

Expressions in NOPEL are broadly categorized as identifiers, constants, structure usage, array usage, and function calls separated by a semi-colon. Parenthesized expressions take on the value of the enclosed expression.

Identifiers

Identifiers are variables that derive their value via the assignment operator, thus making them left-value expressions. In cases where assignment is not performed before evaluation of the variable, some default value is assigned to the identifier.

type identifier = <right-hand value>;

```
int g_globalVar = 10;  // declaration of a variable and an initial value.
trader bob;            // declaration of a variable bob of type trader (defined previously)
int years;             // declaration of a variable with the default value.
```

Constants

Numbers (ints and reals), strings, and dates are all right-value expressions that evaluate to their innate value.

Function calls

A function call is a right-value expression that takes zero or more parameters and returns a declared type or possible void (no value.) The two broad categories of functions are network and local. These are discussed in detail in a later section. In the following example, the beta function is calling the alpha function and assigning the resultant value to the variable s.

type identifier = <right-hand value>;

```
// declaration of a function
function alpha(string name) returns string { return “Hello: “+ name; };
// calling function alpha from function beta.
function beta() returns void { string s = alpha(“Bob”); };
```

Arithmetic

NOPEL supports the mathematical expressions +, -, *, and / for integer and real data types using infix notation. If an integer and real data type are mixed in the expression, the resultant data type is real. The usual order of precedence is in effect, namely *,/ are evaluated before + or -.

The + operator is available as an infix operator between a string and any other NOPEL primitive type. When one of the operands is not a string, the resultant data type is a string regardless of the placement of the non-string operand.

The + operator is also available as an infix operator between a date and an int. The resultant data type is date such that the value of the given date is incremented by the value of the int.

Example usages of arithmetic operators are as shown:

```
int i = 10;
string myString = “This is:”;
string myStringAsPrefix = myNumber+ i; // results in the string “This is 10”
string myStringAsSuffix = i + myString; // results in the string “10This is “
date tomorrow = 23Jan06 +1; // tomorrow has the value 24Jan06
```

Relational

The usual binary operators, “<”, “>”, “<=”, “>=”, “==”, “!=” are available on int and real data types as an infix operator. When comparing objects of different types, int’s are converted to real’s. The resultant value is a boolean.

~~Equal (==) and not equal (!=) can be applied as infix operators on any data type. When 2 objects of different types are compared, the resultant value is always false for “==” and true for “!=”. When 2 objects are of the same type, if the value is the same then the result is true for “==” and false for “!=”. With structure objects, if all the fields have the same values then the result is also true for “==” and false for “!=”~~

Logical

The logical operators “!” (not,) “&” (and,) and “|” (or) are available as infix operators when comparing expressions that result in a boolean value. The order of precedence from high to low is “!”, “&”, “|”.

Basic Statements

The types of statements in NOPEL are assignment, if-then-else, while loops and functions. The first three categories are very similar in syntax to C or Java and are addressed in this section. Functions are addressed in the following section as they are the heart and soul of NOPL.

Assignment

Identifiers, or generically left-value expressions take on a value through the assignment operator “=”. Casting is not available in the NOPEL language so only like values on the left and right side of the “=” are allowed.

```
type identifier = expression;
```

If-Then-Else Conditionals

If-Then-Else conditionals determine the flow of control based on a logical expression. The first execution path is entered if the expression evaluates to true otherwise if the second case “else” is specified, that execution path is entered.

```
if ( logical )
{
        (expression; | statement )*
```

```
} else {
        (expression; | statement )*
}
```

In the following example "out" has the value of 9 at the end of the execution of the conditional. If the value of "in" had any other value than 10, out would have the value 11.

```
int in = 10;
int out = 0;
if ( 10 == in )
{
  out = 9;
} else {
  out =11;
}
```

While Loops

For simplicity, there is a single iterative statement "while" that repeats a given set of expressions and statements until some condition is no longer true. The format for a while loop is:

```
while ( conditional )
{
        (expression | statement | break | continue)*
}
```

The reserved words break and continue have the following effect when placed between { } in a while statement:

- **break --** causes the while loop to exit and execution to continue after the }
- **continue –** causes the execution of the loop to stop at this point and resume after the { in the while declaration.

Functions

The two classifications for functions, local and networked, share the same syntax apart from the keyword *network.* Networked functions are available in a remote context, i.e. available for usage over a network when ultimately compiled into a Java program.

Given the special nature of functions marked *network*, no other function, local or remote may reference these functions. To promote reuse in the language however, networked functions may reference any local function.

Each function declaration begins with the keyword function, the name of the function, a parenthesized list of 0 or more arguments, the keyword *returns* and the return type, possibly *void.* The implementation of the function begins with "{"and is terminated with "}"

Local

The syntax for a local function is generalized as:

```
function identifier(type identifier | type identifier (,type identifier)* | )
```

```
        returns (type | void)
{
        (expression; | statement;)*
        return (identifier | constant | );
}
```

An example implementation of a function is shown here.

```
// a local function
function getAgeLocal( date _birthDate ) returns int {
                // implementation
}
```

Networked

A network function is identical in syntax to a local function with the addition of the keyword *network*.

```
network function identifier(type identifier | type identifier (,type identifier)* | )
        returns (type | void)
{
        (expression; | statement;)*
        return (identifier | constant | );
}
```

An example implementation of a network function is as shown here.

```
// networked function that calls a local function
network function getAge( date _birthDate ) returns int {
                retrun getAgeLocal(_birthDate);
}
```

Servers

The top level construct in NOPEL is a Server. A server consists of zero or more identifiers that may be initialized (assignment statements) and are visible in any declared local or network function.

```
server
{
        (identifiers | assignment statment)*
        (local functions; | network functions)*

}
```

Additional functions

Most languages allow the use of some access to system out. NOPEL uses categorized logging based on the priority of the message. The basic implementation will print out to the console when the application is run. In the future, configurations should allow the redirection of the output to files, sockets, etc. and filtering based on the criteria. The syntax for logging is as shown. Further, the priority of the logging message is listed from least important to highest importance.

(LOG_DEBUG | LOG_INFO | LOG_WARN | LOG_ERROR | LOG_FATAL) **string ;**

In this example the first logging message prints out the in parameter before the execution of the local function and the second logging message prints the result of the local function.

```
network function getAge( date _birthDate ) returns int {
                LOG_INFO ("starting getAge with param: " + _birthDate);
                int out = getAgeLocal(_birthDate);
                LOG_INFO ("ending getAge with result: " + out);
                retrun out;
}
```

Project Plan

Working full time, I mostly worked on the project 2 hours a day and over weekends. The detailed plan is included as a separate Microsoft Office Project plan.

High Level

The high level categories of work are given as follows:

Category	Duration	Start	End
Learning	**3 days**	**23-Jan-06**	**25-Jan-06**
LRM	**2 days**	**26-Jan-06**	**27-Jan-06**
Lexer	**8 days**	**1-Feb-06**	**10-Feb-06**
Parser	**12 days**	**13-Feb-06**	**28-Feb-06**
Semantics (TreeParser)	**18 days**	**1-Mar-06**	**24-Mar-06**
Basic Code Generation	**10 days**	**27-Mar-06**	**7-Apr-06**
Server Side Code Generation	**5 days**	**10-Apr-06**	**14-Apr-06**
Client Side Code Generation	**6 days**	**17-Apr-06**	**24-Apr-06**
General Testing	**1 day**	**25-Apr-06**	**25-Apr-06**
Final Presentation	**2 days**	**26-Apr-06**	**27-Apr-06**

Detailed Plan

Breaking out according to each category:

Task	Duration	Start	End
Learning			
Play with ANTLR Tutorials	1 day	23-Jan-06	23-Jan-06
Write white paper	2 days	24-Jan-06	25-Jan-06
LRM			
Draft LRM	1 day	26-Jan-06	26-Jan-06
Final LRM	1 day	27-Jan-06	27-Jan-06
Basic Prototype	2 days	30-Jan-06	31-Jan-06
Lexer			
Write basic Token parsing	3 days	1-Feb-06	3-Feb-06
Implement Lexer with Error Handling	2 days	6-Feb-06	7-Feb-06
JUnit Test	3 days	8-Feb-06	10-Feb-06
Parser			
Write basic parsing	3 days	13-Feb-06	15-Feb-06

Implement Parser with Error Handling	2 days	16-Feb-06	17-Feb-06
Implement Parser with own AST	2 days	20-Feb-06	21-Feb-06
Refactor Lexer as appropriate	2 days	22-Feb-06	23-Feb-06
JUnit Tests	3 days	24-Feb-06	28-Feb-06
Semantics (TreeParser)			
ANTLR related g files	3 days	1-Mar-06	3-Mar-06
Symbol Table	3 days	6-Mar-06	8-Mar-06
Java helpers for Type definition	3 days	9-Mar-06	13-Mar-06
Exception Classes	2 days	14-Mar-06	15-Mar-06
Exception Handling	2 days	16-Mar-06	17-Mar-06
Refactor Lexer and Parser as necessary	2 days	20-Mar-06	21-Mar-06
JUnit Tests	3 days	22-Mar-06	24-Mar-06
Basic Code Generation			
ANTLR related g files	3 days	27-Mar-06	29-Mar-06
Server level classes	3 days	30-Mar-06	3-Apr-06
Type Generation	1 day	4-Apr-06	4-Apr-06
Statement Generation	1 day	5-Apr-06	5-Apr-06
Function Generation	1 day	6-Apr-06	6-Apr-06
Types in side functions	1 day	7-Apr-06	7-Apr-06
Server Side Code Generation			
Statements in functions	1 day	10-Apr-06	10-Apr-06
Write out file no sockets	1 day	11-Apr-06	11-Apr-06
Write files with socket interaction	1 day	12-Apr-06	12-Apr-06
Test with dummy driver	1 day	13-Apr-06	13-Apr-06
Refactor as necessary	1 day	14-Apr-06	14-Apr-06
Client Side Code Generation			
functions no params no return	1 day	17-Apr-06	17-Apr-06
functions with parameters (primitives)	1 day	18-Apr-06	18-Apr-06
functions with params and return (primitives)	1 day	19-Apr-06	19-Apr-06
functions with arrays	1 day	20-Apr-06	20-Apr-06
functions with structs	1 day	21-Apr-06	21-Apr-06
functions with array of structs	1 day	24-Apr-06	24-Apr-06
General Testing			
Final Presentation			
Draft document	1 day	26-Apr-06	26-Apr-06
Final Iteration	1 day	27-Apr-06	27-Apr-06

Roles and Responsibilities

Dan did all the work. The other members of the team were slackers. Ok, they did not exist but all the same, slackers.

Development Environment

Development took place on a Windows XP operating system using Eclipse 3.1 with Java 1.5. To aid in developmement/debugging I bought the 3rd party vendor software Antlr Studio by Placid Systems. (http://www.placidsystems.com/) The most useful feature of

the software was the color coding of the g files to easily identify java sections, comments, tokens, etc.

Style Guide

NOPEL style is designed using coding standards set forth at Goldman Sachs for Java Development and is as follows:

Comments

- Block comments (multi-line comments) should be used to separate major functional portions of code. Note that there are always two blank lines above and one blank line below a block comment. Also note that the actual comment text is indented by a space from the block.
- Single line comments can be done with a `//`-type comment.

Function Calls

- An open-parenthesis always follows the method name or Java keyword without any intervening white space.
- There is never white-space before or after a semicolon.
- An open-parenthesis is always followed by exactly one space. A closing-parenthesis is always preceded by exactly one space. A comma is always followed by exactly one space.
- Any function arguments that overflow onto another line should be indented by 8 spaces.

If statements are formatted as follows:

- Single operations do not have curly braces.
- `else if` and `else` are at the same indentation level as `if`.
- There are no blank lines after the opening curly braces.
- There are no blank lines before the closing curly braces.
- An overflowed line should be indented 8 spaces or logically and readably lined up with the previous line.
- Logical connectors (`&&` and `||`) should be placed at the start of an overflowed line.
- Nested parens [and square brackets] are grouped together with no space in between - `))` rather than `) )`.

For statements are formatted as follows:

```
    for( int i = 0; i < SOME_VALUE; i++ )
        // Single operation goes here

    for( int i = 0; i < SOME_VALUE; i++ )
    {
        // Multiple operations go here
    }
```

While statements are formatted as follows:

```
    while( i != SOME_VALUE )
        // Single operation goes here

    while( i != SOME_VALUE )
    {
        // Multiple operations go here
}
```

Switch statements are formatted as follows:

```
    switch( var )
    {
        case ONE_CASE:
            // Operations go here
            break;

        case ANOTHER_CASE:
        case ONE_MORE_CASE:
        {
            int blockLocalVariable;


            // Operations go here
            break;
        }

        case YET_ANOTHER_CASE:
            // Operations go here
            /* FALL THROUGH */

        default:
            // Operations go here
            break;
    }
```

Variable declarations

Sometimes in large blocks it is reasonable to mix variable declarations with statements. Try to retain alignment and reasonable inclusion of blank lines.

Use of exceptions

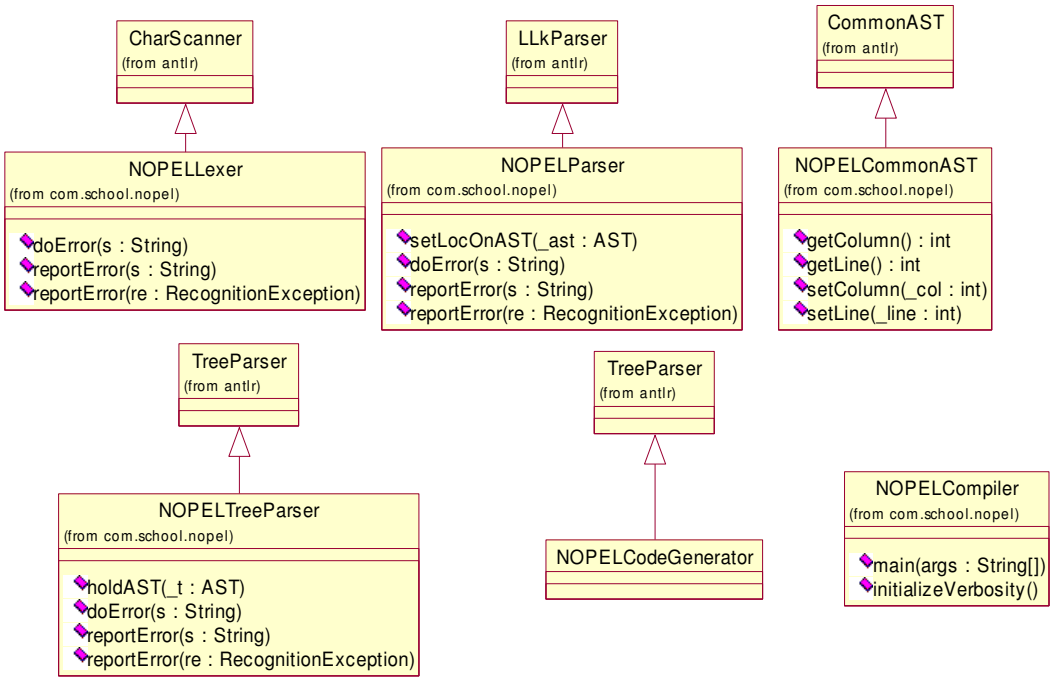
Exceptions should only be used to signal serious, out of band conditions from which recovery is either not obvious or is up to the caller. Exceptions should not be thrown and caught as part of the normal path of execution; rather, conditions should be checked first.

Architectural Design

Antlr Components

The Lexer, Parser and TreeParser implement a common paradigm for error handling by overwriting the *reportError* methods and delegating to a *doError* method. The NOPEL language incorporates its own AST, **NOPELCommonAST** in the **NOPELParser** and **NOPELTreeParser** to report the line number and column number on which a semantic error occurred. Other than these features, explained in detail below, the implementation followed the standard implementation found in previous projects.

The **NOPELCodeGenerator** did not incorporate the concepts of error handling or a special AST since it is assumed that at this point in the compilation process there is no possibility of error. If time permitted I would implement an optimizer component in the same manner.



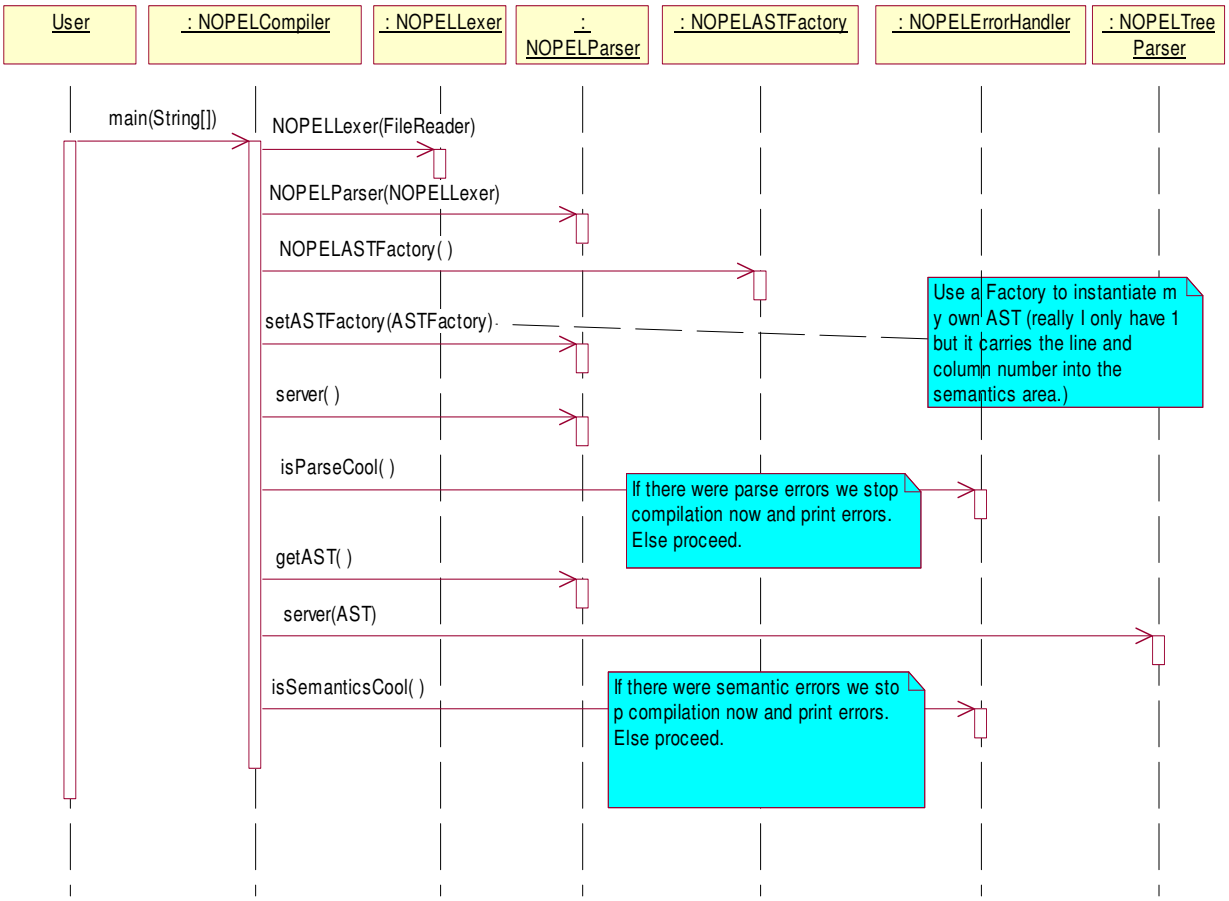
Compilation Process

The compilation of a NOPEL program consists of the following steps:

1. Parsing
2. Semantics Checking
3. Server side code generation
4. Client side code generation.

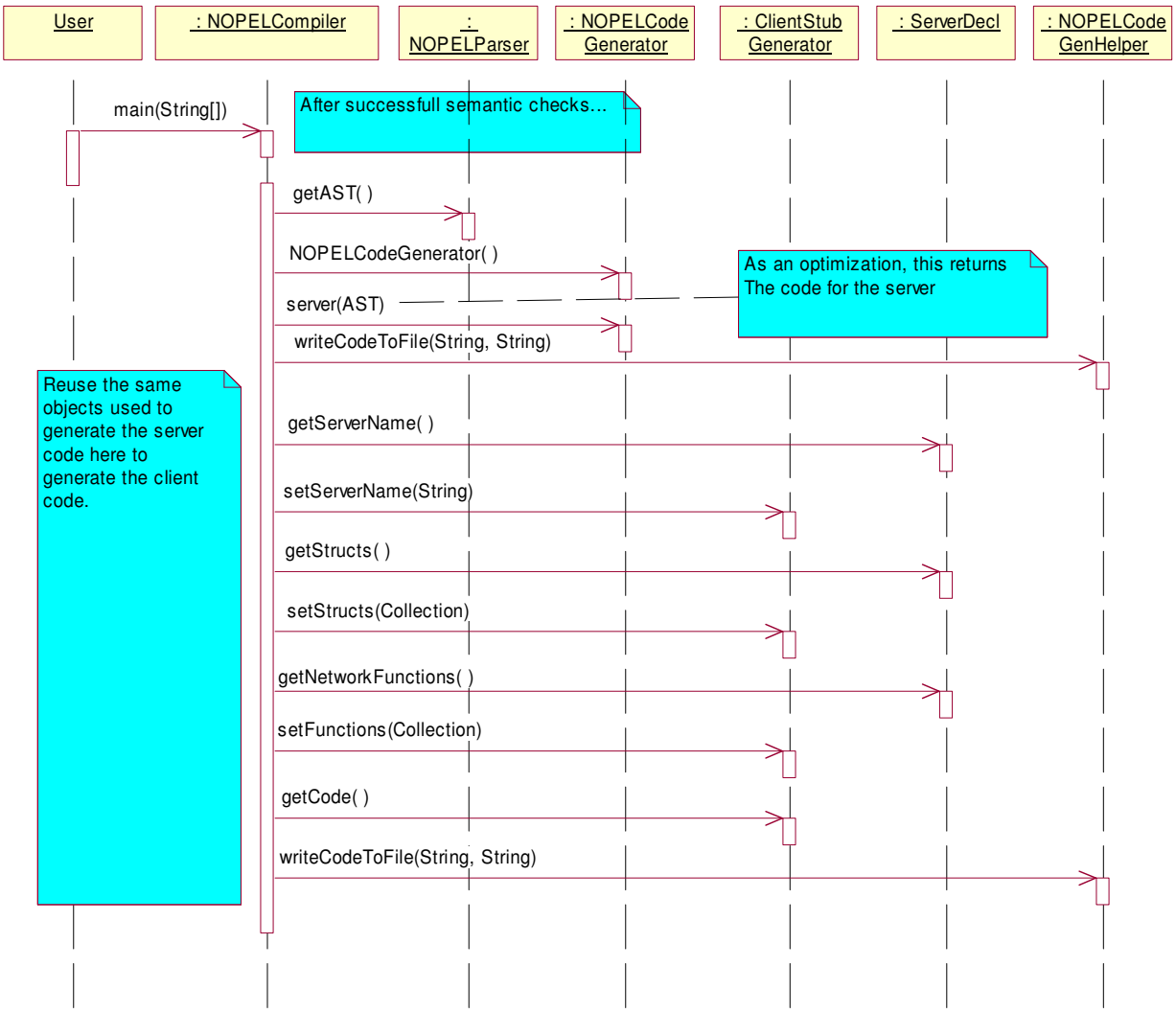
All of the steps are performed in the main method of the NOPELCompiler.java file but for clarity I will break it out into the front end and back end.

Front End (Parsing/Semantics)



Back End (Server and Client Code Generation)

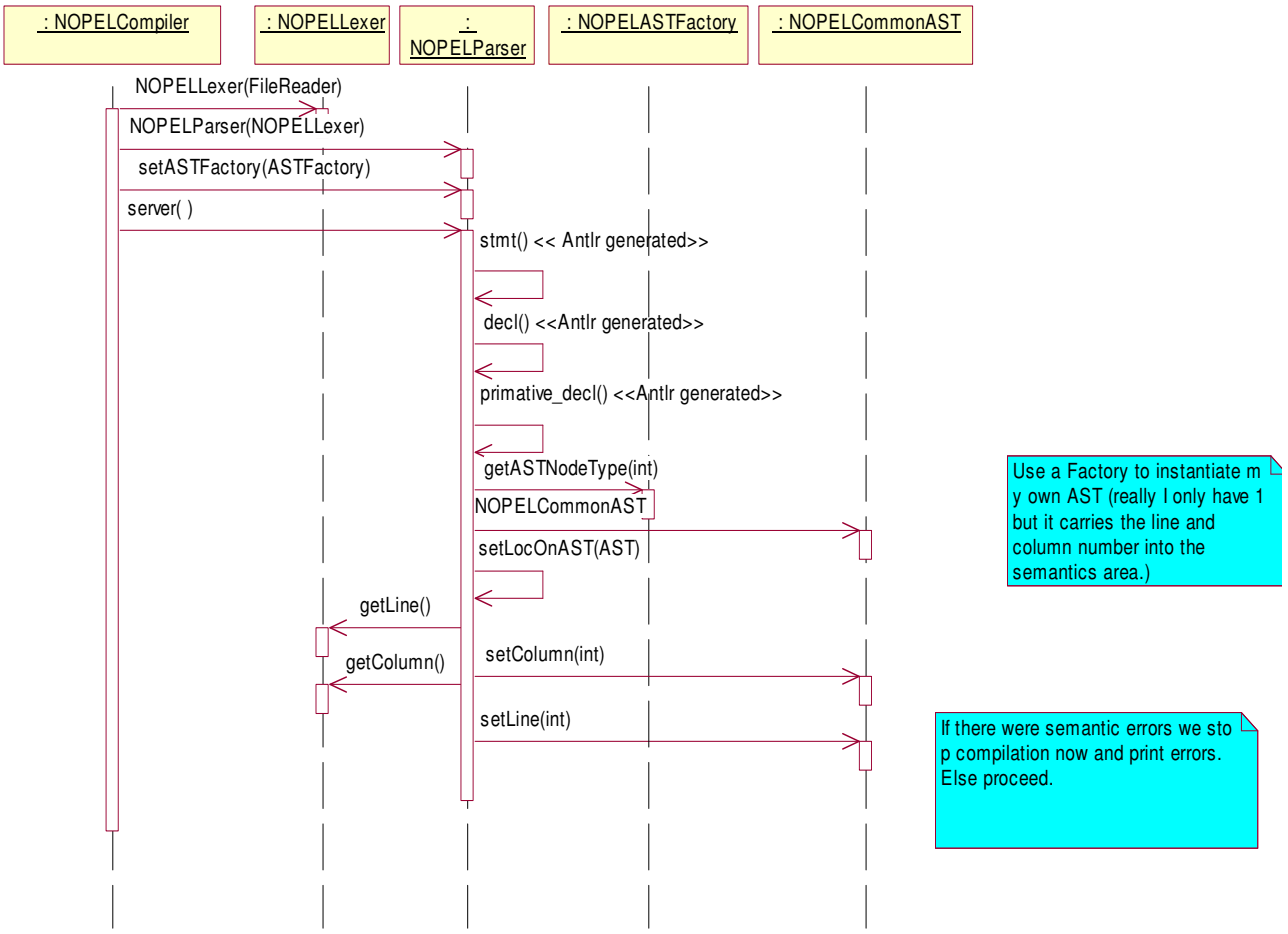
The Antlr file **NOPELCodeGenerator** populates various java helper classes, namely **ServerDecl**, with details it can use to generate code. E.g. for a **PrimativeDecl,** the code generator instantiates the object and sets as properties the id for the variable, the type, and the value. By instantiating these objects and not writing code directly into some buffer, the **NOPELCompiler** can reuse these instances to construct both the server and the client code. With an eye to the future, these classes can be easily modified to generate code in other object oriented languages. Before delving into the details of the compilation process, here is the rest of the interaction diagram for the NOPELCompiler.



Parsing

This interaction diagram shows how the NOPELParser handles parsing a program that has a single integer declaration. It is important to note the use of the **NOPELCommonAST** and factory that carry the most recent line and column number into the semantic analysis phase. When performing a simple parse, the Parser gets the line number directly from the Lexer. However, any tree walking has no association to the actual text so this information must be recorded on the AST the Parser generates.

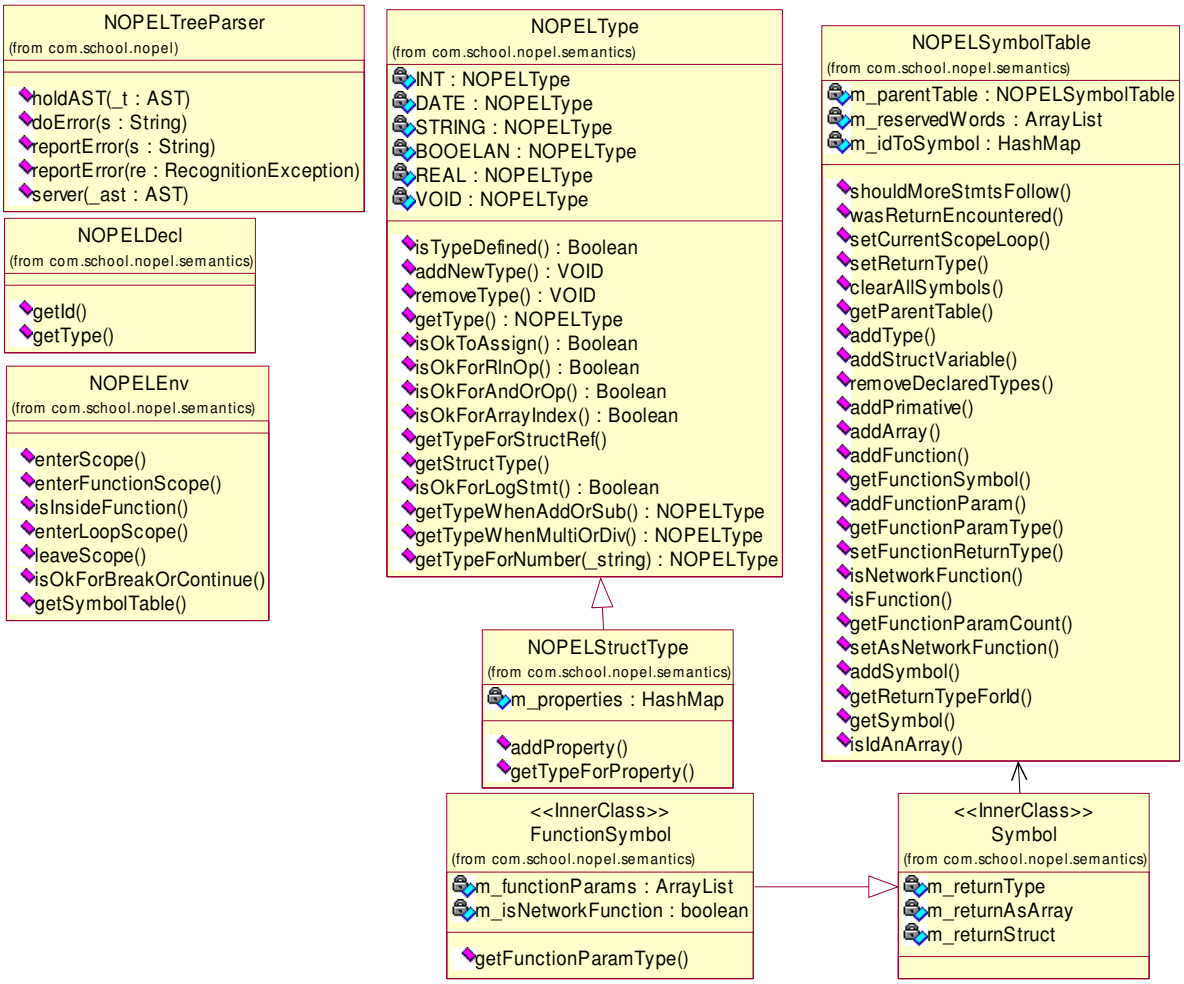
```
server simple {
    int i = 20;
}
```



Semantic Components

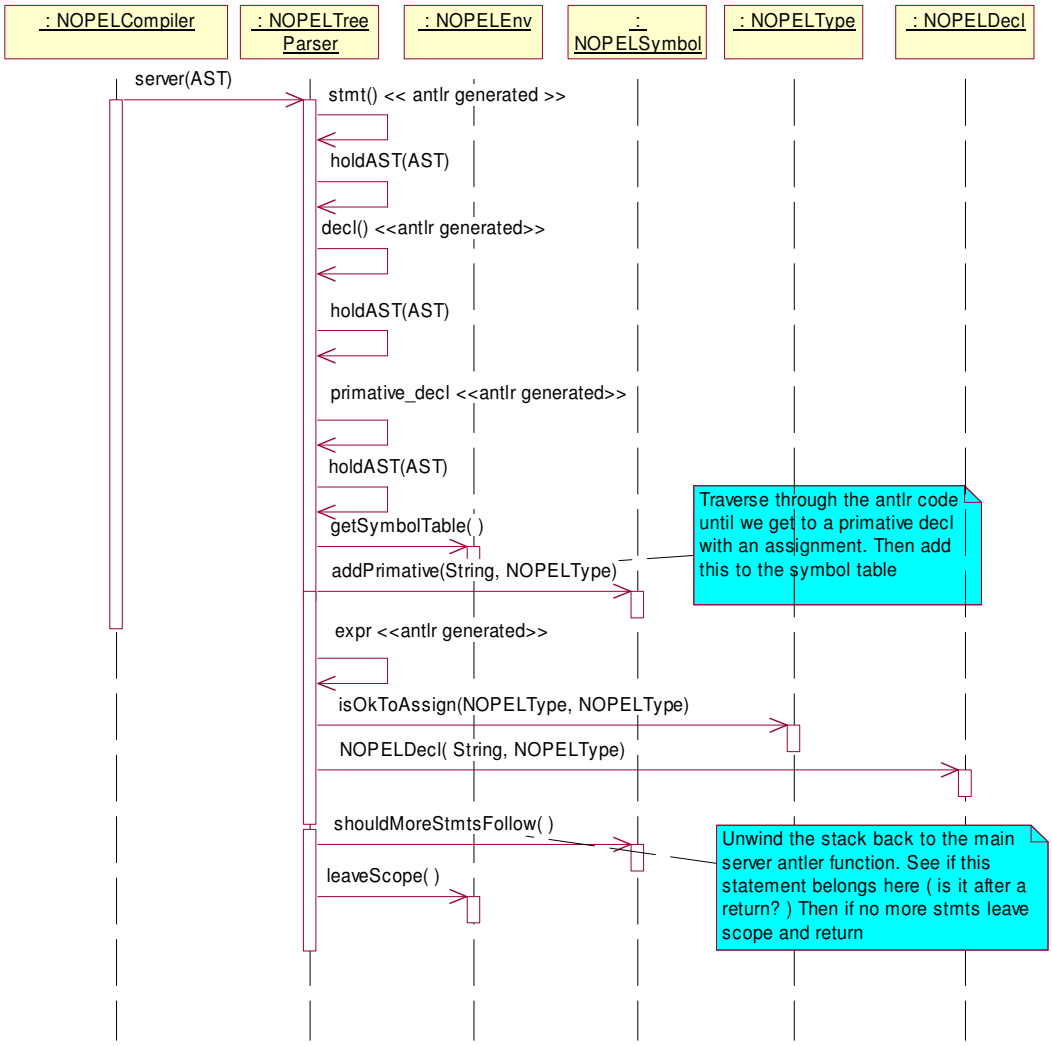
The major components in semantic analysis are shown in the following diagram. The **NOPELSymbolTable** is a wrapper around a dictionary of defined symbols that contains a reference to a parent symbol table when a new scope is entered. New scopes are defined and delineated as function, structure, and loop scope.

The **NOPELType** represents all defined types in NOPEL including structures. The implementation has a number of helper methods to determine the relationship among types in the context of expressions. E.g. *isOkForAndOrOp(…) and isOkToAssign(…).*



Continuing with the previous example, the semantic analysis interaction diagram for this simple server is shown below.

```
server simple {
    int i = 20;
}
```



Error Handling

Error handling is implemented in a similar fashion in both the **NOPELTreeParser** and **NOPELParser**. If an error is encountered at any point in the program, NOPEL records the error in the **NOPELErrorHandler** (singleton) and continues on with the compilation. Once the current phase of compilation is complete (Parsing, Semantic, Code Generation), if any errors were encountered the compiler reports them to the user and the compilation halts. In this manner NOPEL can spot multiple errors but reasonably terminate without generating bogus errors due to failures in the previous phase.

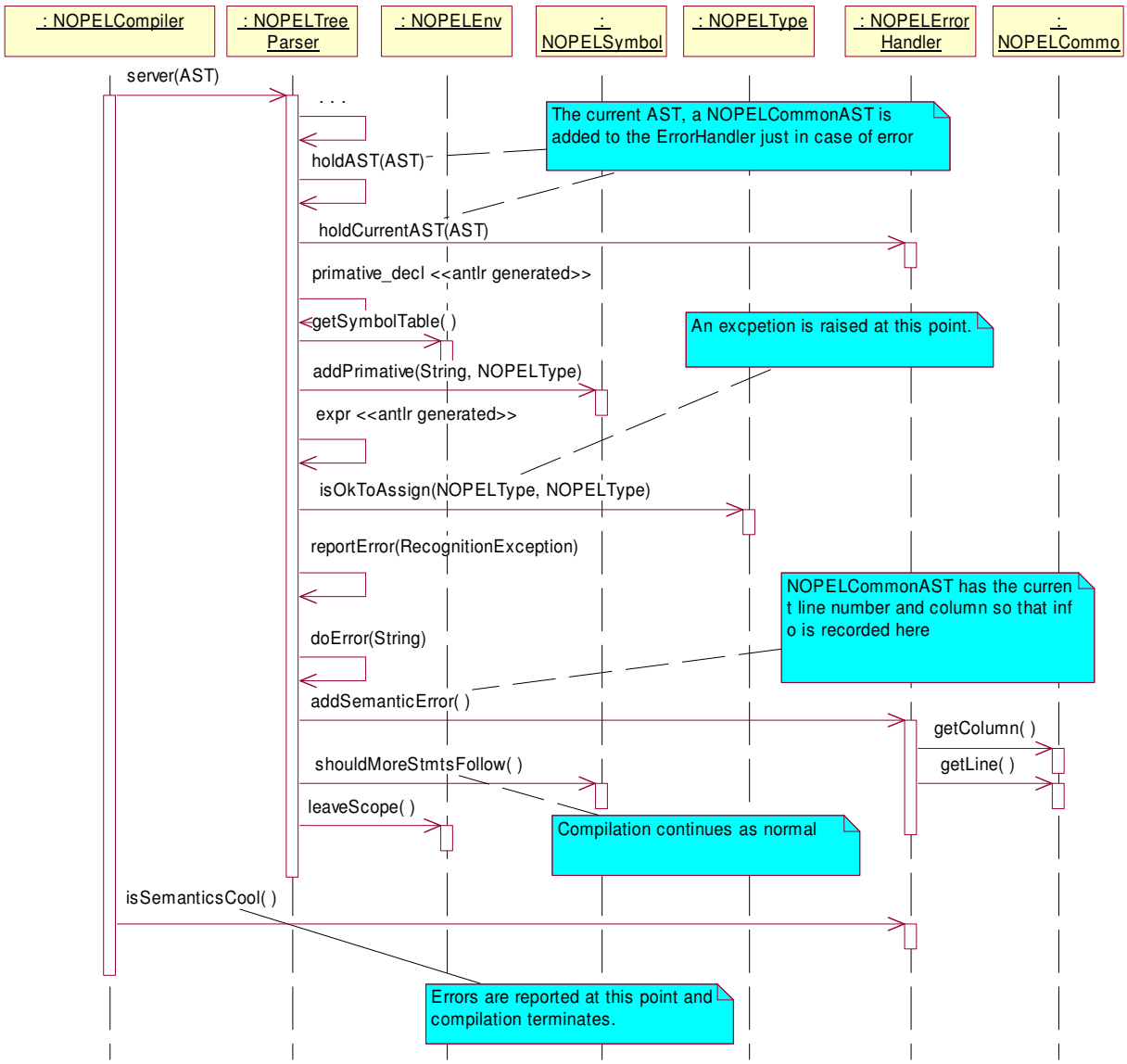
As an example, consider in the previous interaction if the assignment

```
int i = 20;
```

was instead:

```
int i = “some string”;
```

The NOPEL compiler would operate as follows:

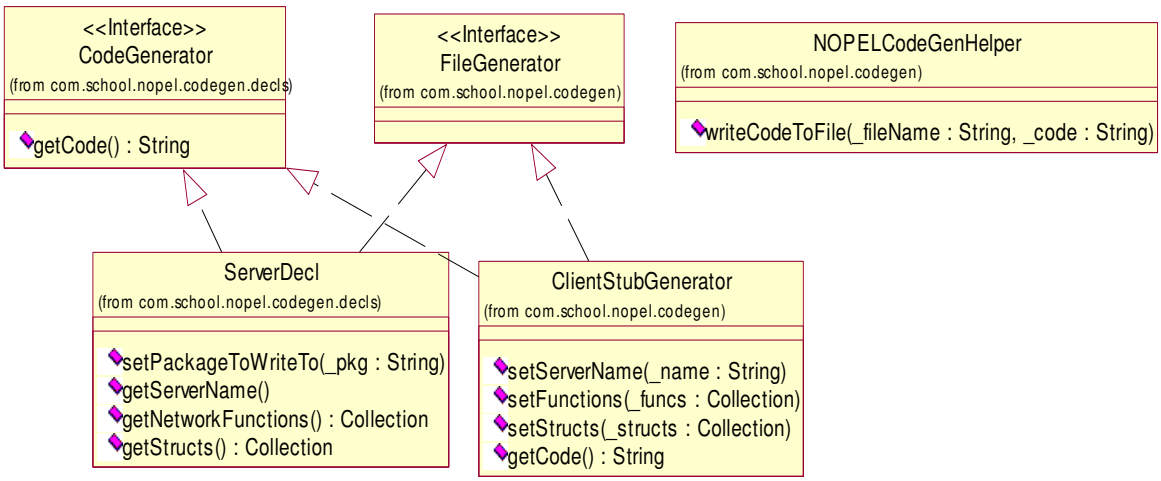


Code Generation Components

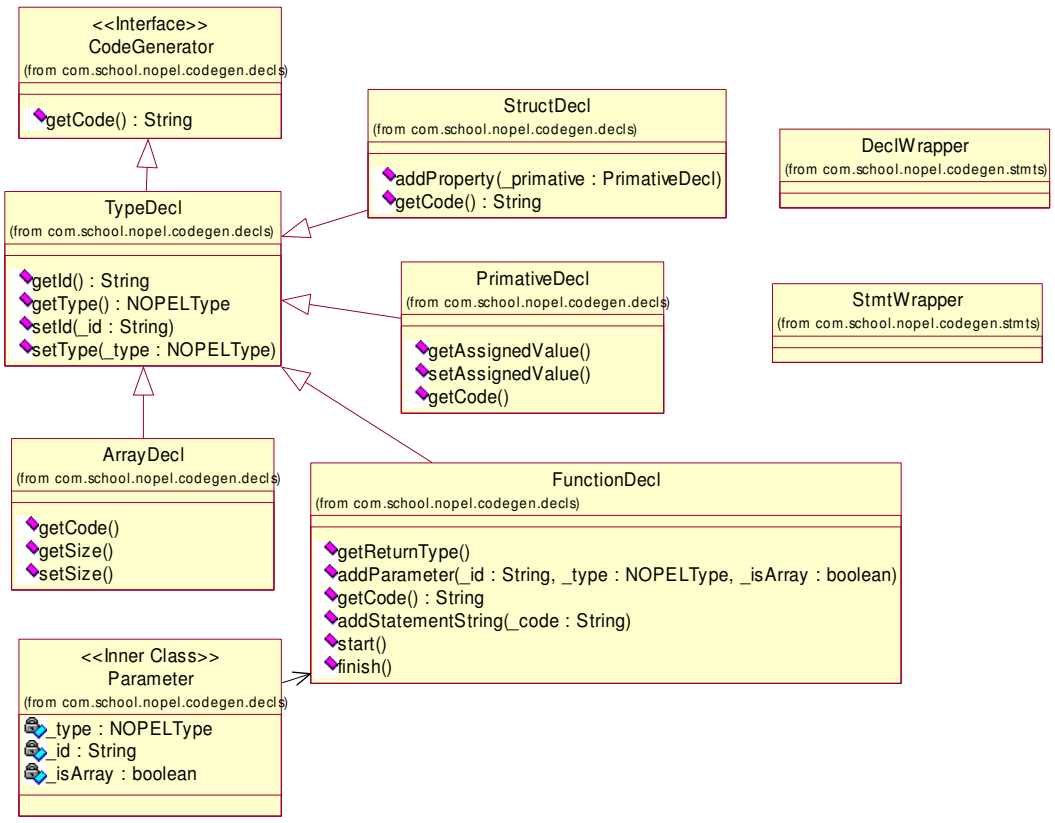
The final phase of the compilation process is the generation of the server and client files. NOPEL utilizes a generated Tree Parser to generate code. The **NOPELCodeGenerator** does not actually generate code, but rather walks the tree and creates objects that know how to generate code. With this design, the same Tree Parser can be used later on to generate code in other languages. E.g. C++ or C#.

The main "workers" of the code generation phase are shown in the following class diagram. The **ServerDecl** class is a glorified collection of **StatementWrapper** and **DeclWrapper** objects that the **CodeGenerator** iterates to generate a string representing the server code. As the **NOPELCodeGenerator** walks the tree, it populates this data structure.

The **ClientStubGenerator** similarly uses the **ServerDecl** to generate the string that represents the client code. The **NOPELCodeGenHelper** is client/server agnostic and writes out the both sets of code to the appropriate files.

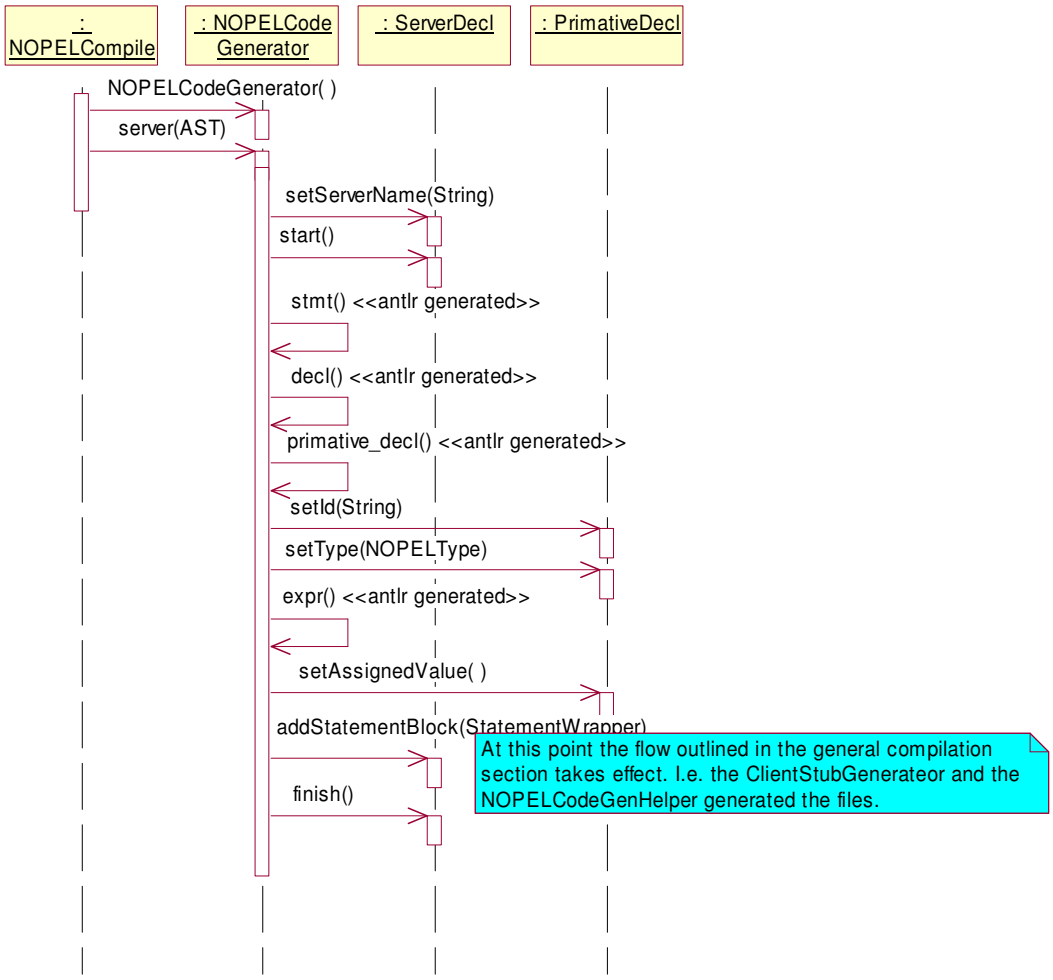


The data structures that contain the implementation of code generation are shown in the following diagram. Each of these structures generates code directly or delegates to other **CodeGenerator** objects within it. E.g. a **FunctionDecl** will have one or more **TypeDecls** and **StmtWrappers** contained within it. The beauty of this design is that the parent **CodeGenerator** classes can determine what components to use. E.g. the **ServerDecl** will use the full implementation of the **FunctionDecl** to generate code while the **ClientStubGenerator** will only use the signature of the **FunctionDecl** to generate code.



The steps in code generation for the trivial example are as shown.

```
server simple {
    int i = 20;
}
```



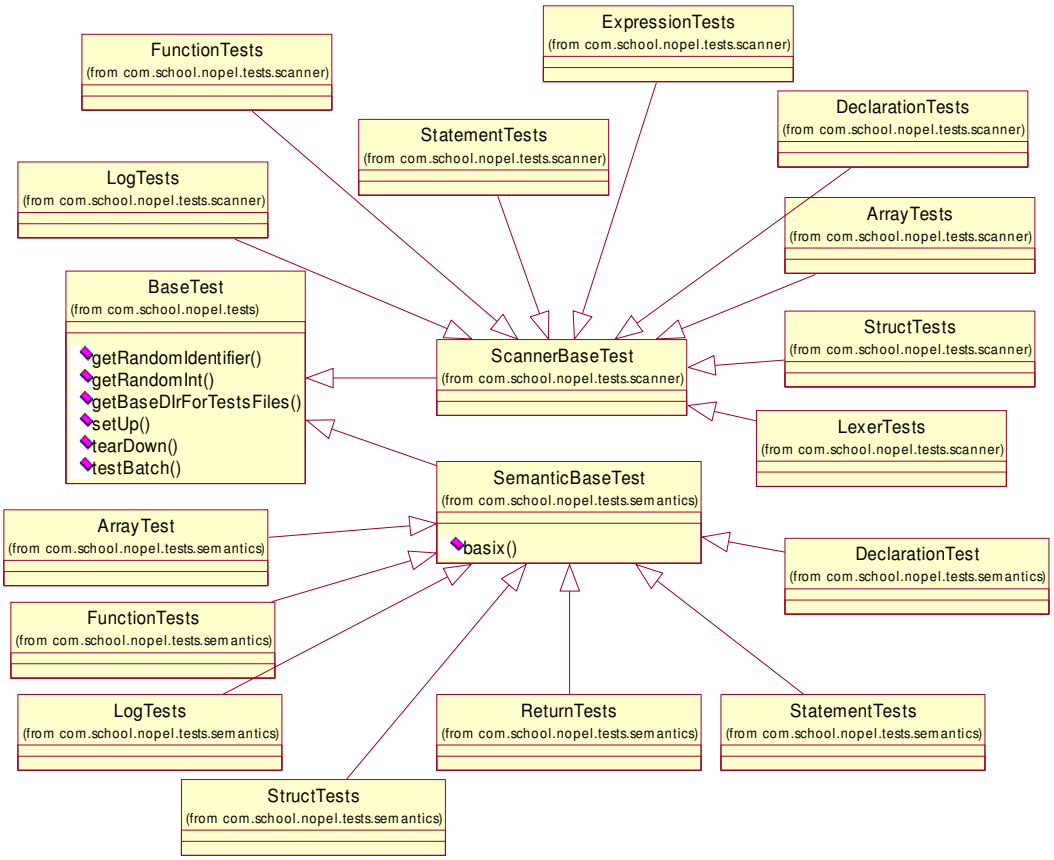
Test Plan

Scanner and Semantic testing for NOPEL is implemented exclusively as JUnit test cases. As I wrote a new part of the lexer/parser I would write two corresponding tests in JUnit. The first test would test out code that should pass muster while the second tested code that should cause an error. As I moved on to writing other components, I repeated this process continually building up a test suite.

I tested out the code generation piece by hand and never had an opportunity to write formal JUnit test cases.

JUnit Test Structure

All tests extended from a common base class that operated as shown.



Unit Test Examples

```
public void testInitWithExpressions()
{
String validBasix[] = {
        "server dans{int d = 10;}",             // INT with constants
        "server dans{int d = 1 + 2;}",          // INT with addition expr
        "server dans{int d=1; int e = d +1;}", // INT with addition expr & var
        "server dans{int d=1; int e = 1 +d;}", // INT with addition expr & var

        "server dans{int d=1; int e = 2; int f = d+e;}",// INT with addi expr & var
        "server dans{int d= 2 * 2;}",                   // INT with multi expr
        "server dans{int d= 2; int g= d* 2;}",          // INT with multi expr & var
        "server dans{int d= 2; int f= 2*d;}",  // INT with multi expr & var
        "server dans{int d=1; int e = 2; int f = d*e;}",// INT with addi expr & var

        // other tests with multiple levels of expressions and parens
        "server dans{int d=1; int e = 2; int f = d*e+2;}",
        "server dans{int d=1; int e = 2; int f = d*e+2*3;}",
        "server dans{int d=1+2+3; int e = 2+d; int f = d*e+2*3;}",
        "server dans{int d=(1+2+3); int e = (2+d); int f = d*(e+2)*3;}",
```

```
        // Test with initialization by function
        "server dans{int d=foo(34);}",
        };
        testBatch(validBasix,true);
}
```

Iterative Testing

As I progressed into other levels of the project (semantics and code generation) I inevitably came upon errors in the previous sections. At that time I would incorporate more tests. E.g. I found that java would often add an “E” to my real types that my parser could not handle. Once I identified and fixed the problem in the parser, I included the following test to catch this in the future in case I reintroduced the bug.

```
/**
 * Test that E in the exponent is not allowed
 */
public void testE()
{
basix("server "+ getRandomIdentifier()+ "{real r=90234234.2344E;}",false);
}
```

Advanced Testing

For even better testing, in many cases I incorporated a *testRandom**** method that would randomly generate test cases. E.g. for declarations, I had a method that would generate a random identifier with a random type then assign an appropriate random value.

```
/**
 * Tests out random declarations
 */
public void testRandomDeclarations()
{
        Random r = new Random();
        int next = 0;
        StringBuffer s = null;
        for (int j=0; j < m_maxRandomToRun; ++j)
        {
                next = Math.abs(r.nextInt()) % m_maxDeclarations;
                s = new StringBuffer("server "+getRandomIdentifier()+" {");

                for (int i=0; i < next; ++i)
                {
                        s.append(BaseTest.declHelper.getRandomDecl(r));
                }
                s.append("\n}");
                basix(s.toString(),true);
```

```
        }
    }

    protected static String getRandomIdentifier()
        {
            // Generate a random string of a random length < 20 to be nice
            int length = getRandomInt(3,20);
            StringBuffer string = new StringBuffer();
            // Add the first one
            int index = getRandomInt(1,alphabet.length()-1);

            string.append(alphabet.charAt(index));
            for (int i=0; i < length; ++i)
            {
                index = getRandomInt(1,alphabetMore.length()-1);

                string.append(alphabetMore.charAt(index));
            }
            return string.toString();
        }
```

Lessons Learned

The main lesson I learned was to not get ahead of myself in implementing the project. I found several times that I would bang my head against a wall trying to figure out how to do something when the next lecture (that I had not yet watched) explained what I was trying to do.

Another lesson was to keep the Language Reference Manual intentionally vague in terms of implementation. I found many times I made claims in the LRM that I tried to force into the implementation but code not. Keep the LRM as a specification and iron out the implementation details later. E.g. in my LRM I claimed that I would implement the socket communication as UDP. When implementing I found it easier to use TCP. In the context of the LRM this is irrelevant information anyways.

Appendix Client Configuration

If running on the CLIC machines, a SAX implementation is already on the classpath so there is nothing to do.

However, running in some other environment, you must have a SAX implementation on your classpath to run the client. I used crimson and have included it in the NOPELCompiler jar file. To use it, unpack crimson.jar and when running your client stub, ensure you run with the java env variable: -Dorg.xml.sax.driver=org.apache.crimson.parser.XMLReaderImpl set.

Appendix A NOPELParser.g

```
header {

	package com.school.nopel;
	import com.school.nopel.semantics.*;
	import com.school.nopel.compiler.NOPELErrorHandler;
}
class NOPELLexer extends Lexer;
options{
	k = 3; // need 3 so I can see through dates
	charVocabulary = '\3'..'\377';
	testLiterals = false;
	exportVocab = NOPELVocab;
}
{
	static java.util.logging.Logger m_logger =
java.util.logging.Logger.getLogger("Lexer");
	public void doError( String s )
	{
		if (null != s)

	NOPELErrorHandler.addParseError(getFilename(),getLine(),getColumn(),s );

	}
	public void reportError( RecognitionException re )
	{
		doError( re.getMessage() );
	}
	public void reportError( String e )
	{
		doError( e );
	}
}
// -- Basics
protected
ESC
	: '\\'
	(	't'
	|	'n'
	|	'b'
	|	'\\'
	|	'"'
	)
	;
protected
ALPHA : 'a'..'z' | 'A'..'Z';
protected
DIGIT : '0'..'9';

// LRM 1.1   Tokens
WS
	: (	' '
	|	'\t'
	|	'\n'	{ newline(); }
	|	'\r' ('\n')? { newline(); }
	)
	{$setType(Token.SKIP);}
	;
// LRM 1.1.1 Comments
MULTI_LN_CMT
	: "/*" (
	options {
	greedy=false;
	generateAmbigWarnings=false;
}
	:	'\r' '\n' {newline();}
	|	'\r'  {newline();}
	|	'\n'	{newline();}
	|	~('\n'|'\r')
	)*
```

```
        "*/"
        {$setType(Token.SKIP);}
        ;
// LRM 1.1.1 Comments
SINGLE_LN_CMT
        : "//"
        (~('\n'|'\r'))* (('\n'|'\r'('\n')?){newline();})?
        {$setType(Token.SKIP);}
        ;
// LRM 1.1.2 Identifiers
ID options { testLiterals = true; }
        : (ALPHA | '_')(ALPHA | DIGIT | '_')* ;

// LRM 1.1.3 Keywords Skip for now.

// LRM 1.1.4 Numbers Ints and Reals
// So we can use signed numbers later these are here for ref only
protected INT : (DIGIT)+;
protected REAL : (DIGIT)+ DOT (DIGIT)+;

NUMBER
        : (DIGIT)+
                ( (DOT (DIGIT)+) { $setType(REAL); }
                | /* empty */ { $setType(INT); }
)
;
// LRM 1.1.5 Strings
STRING
        : '\"' (ESC|~'\"')* '\"';

protected DAY               : DIGIT DIGIT;
protected YEAR              : DIGIT DIGIT DIGIT DIGIT;
protected MONTH     : ALPHA ALPHA ALPHA;
// LRM 1.1.6 Dates
DATE
        :
        DAY MONTH YEAR
        {
                // We need to test that this is properly formatted.
                try {
                        java.text.SimpleDateFormat sdf = new
java.text.SimpleDateFormat("ddMMMyyyy");
                        java.util.Date dIn = sdf.parse(getText());
                        String dOut = sdf.format(dIn);
                        if (!dOut.toUpperCase().equals(getText().toUpperCase()))
                        {
                                // Raise an exception here
                                doError(getText() +" did not match a date formatted
as ddMMMyyyy");
                                throw new RecognitionException("Exception matching: "
+ getText() +" as a date ddMMMyyy");
                        }
                } catch (Exception e)
                {
                        // Raise an exception here
                        //doError("Exception matching: " + getText() +" as a date
ddMMMyyy"+ e.getMessage());
                        throw new RecognitionException("Exception matching: " +
getText() +" as a date ddMMMyyy"+ e.getMessage());
                };
        };


// LRM 1.1.7 Convenience Tokens
LCURLY : '{';
RCURLY : '}';
LPAREN : '(';
RPAREN : ')';
LSQR : '[';
RSQR : ']';
SEMI : ';';
// -- didnt mention these in LRM but nice to haves
```

```
ASSIGN : '=';
DOT : '.';
MULT : '*';
PLUS : '+';
MINUS : '-';
DIV : '/';
GE : ">=";
LE : "<=";
GT : '>';
LT : '<';
EQ : "==";
NEQ : "!=";
AND : "&";
OR : "|";
NOT : "!";
COMMA : ",";

class NOPELParser extends Parser;
options{
	k = 3; // We need this to see arrays
	buildAST = true;
	exportVocab = NOPELVocab;
}
tokens {
	NEGINTREF;
	NEGREALREF;

	RETURNSTMT;
	IFSTMT;
	WHILESTMT;
	FUNCCALLEXPR;
	FUNCCALLPARAMS;
	FUNCDECLPARAMS;
	FUNCDECL;
	NETWORKFUNCDECL;

	STRUCTFIELDS;

	SERVER_AST;
	GLOABLDECLS;

	ARRAY_DECL;
	STRUCT_DECL;
	PRIMATIVE_DECL;
}
{
	static java.util.logging.Logger m_logger =
java.util.logging.Logger.getLogger("Parser");
	public void setNOPELASTFactory(antlr.ASTFactory _fact)
	{
		astFactory = _fact;
	}
	/*
	 * Maybe I missed it, but it would be nice if Antler used a factory to
get the AST instances
	 * then this would be a little cleaner...
	 */
	public void setLocOnAST(AST _ast)
	{
		if (null != _ast) // is null when guessing. This is ok
		{
			antlr.LexerSharedInputState ls =
((NOPELLexer)getInputState().getInput().getInput()).getInputState();

			((NOPELCommonAST)_ast).setLine(ls.getLine());
			((NOPELCommonAST)_ast).setColumn(ls.getColumn());
		}
	}
	public void doError( String s )
	{
		if (null != s)
		{
```

```
                    antlr.LexerSharedInputState ls =
((NOPELLexer)getInputState().getInput().getInput()).getInputState();

      NOPELErrorHandler.addParseError(ls.getFilename(),ls.getLine(),ls.getColum
n(),s );
            }
      }
      public void reportError( String s )
      {
            doError( s );
      }
      public void reportError( RecognitionException re )
      {
            doError( re.getMessage() );
      }
}
signedNumber : (MINUS NUMBER^) |NUMBER;
// LRM 1.2.1.1      Primitive Types
// This is one option or I can check later on.
type_name : ("boolean"| "date" | "string" |"int" | "real")
{setLocOnAST((AST)currentAST.root);}
;

boolean_value : ("true" | "false")
{setLocOnAST((AST)currentAST.root);}
;

const_value : (boolean_value | signedNumber | STRING | DATE)
{setLocOnAST((AST)currentAST.root);};
//primative_decl : type_name ID (ASSIGN^ expr)? SEMI!
primative_decl : type_name ID (ASSIGN expr)? SEMI! // dont tree the assign
semantic chqr will be easier
{ #primative_decl=#([PRIMATIVE_DECL,"primative_decl"],primative_decl);}
{setLocOnAST((AST)currentAST.root);}
;

//LRM 1.2.1.2 Structures // N.B. change from LRM I need an ID to name the
struct.
struct_fields : (primative_decl)*
{ #struct_fields = #([STRUCTFIELDS, "structFields"], #struct_fields); }
{setLocOnAST((AST)currentAST.root);}
;
// This is the declaration for the type not the instantiation of a struct
variable.
struct_decl :
      "struct"!
      ID LCURLY!
      struct_fields
      //(primative_decl)*
      RCURLY! SEMI!
{ #struct_decl=#([STRUCT_DECL,"struct_decl"],struct_decl);}
{setLocOnAST((AST)currentAST.root);}
      ;

// LRM 1.2.1.3      Arrays
// changing the spec to allow array of structs
array_decl : (type_name | ID) ID LSQR! size:INT
{
      try {
            int value = Integer.parseInt(size.getText());
            if (value < 1)
            {
                  doError("Array declaration cannot have length < 1. This
was: "+value);
                  throw new RecognitionException("Array declaration cannot
have length < 1. This was: "+value);
            }
      } catch (Exception e)
            {
                  doError("Problem checking if array size > 0");
                  throw new RecognitionException("Problem checking if array
size > 0" + e.getMessage());
```

```
                }
}
                        RSQR! SEMI!
{ #array_decl=#([ARRAY_DECL,"array_decl"],array_decl);}
{setLocOnAST((AST)currentAST.root);}
;
// Group for convenience
decl : (primative_decl | array_decl| struct_decl )
{setLocOnAST((AST)currentAST.root);}
;

// LRM 1.2.2.1      Identifiers
// An element of an arry, a struct, or a property in a struct.
// or an array of structs, referencing a property in the struct.
varRef: ID( (LSQR expr RSQR(DOT ID)?) | (DOT (ID | "length")) )?
{setLocOnAST((AST)currentAST.root);}
;

// LRM 1.2.2.2      Constants
booleanRef : "true" | "false";
//constRef : INT | REAL| STRING | DATE | booleanRef;
negIntRef: (MINUS! INT)
{ #negIntRef = #([NEGINTREF, "negIntRef"], #negIntRef); }
{setLocOnAST((AST)currentAST.root);}
;
negRealRef: (MINUS! REAL)
{ #negRealRef = #([NEGREALREF, "negRealRef"], #negRealRef); }
{setLocOnAST((AST)currentAST.root);}
;
constRef
:
                        (STRING             |
                        DATE            |
                        booleanRef      |
                        INT                 |
                        REAL            |
                        negIntRef       |
                        negRealRef)
{setLocOnAST((AST)currentAST.root);}
              ;

//constRef : signedNumber | STRING | DATE | booleanRef;

// EXPRESSIONS
basicExpr :
       ((ID LPAREN) => funcCallExpr |// If we see ID then LParen assume this is
a func call
       varRef |
       constRef |
       LPAREN! expr RPAREN!)
       {setLocOnAST((AST)currentAST.root);}
;
multiExpr : (basicExpr ((MULT^|DIV^) basicExpr)*)
{setLocOnAST((AST)currentAST.root);};
// LRM 1.2.2.4      Arithmetic
addExpr : (multiExpr ((PLUS^ | MINUS^) multiExpr)*)
{setLocOnAST((AST)currentAST.root);};
// LRM 1.2.2.5      Relational
relationalExpr : (addExpr ((GE^|LE^|GT^|LT^|EQ^|NEQ^) addExpr)*)
{setLocOnAST((AST)currentAST.root);}
;


negExpr : ((NOT^)* relationalExpr)
{setLocOnAST((AST)currentAST.root);};
expr : (negExpr ((AND^|OR^) negExpr)*)
{setLocOnAST((AST)currentAST.root);}
;

breakOrContinue : (("break" | "continue") SEMI!)
{setLocOnAST((AST)currentAST.root);};
```

```
// STATEMENTS
stmt : (
              (ID ID SEMI!) => structInstantiation |
              decl                  |
              logStmt                     |
              assignStmt            |
              ifStmt                      |
              whileStmt             |
              funcCallStmt  |
              returnStmt            |
              SEMI!                 |
              breakOrContinue
              )
{setLocOnAST((AST)currentAST.root);}
  ;


/* TODO This was not in the spec me thinks. Check */
structInstantiation : (ID ID SEMI!)
{setLocOnAST((AST)currentAST.root);};
// LRM 1.2.3.1      Assignment
assignStmt: (varRef ASSIGN^ (expr) SEMI!)
{setLocOnAST((AST)currentAST.root);};
// LRM 1.2.3.2      If-Then-Else Conditionals
ifStmt : "if" LPAREN! expr RPAREN!
              LCURLY!
        (stmt)*
              RCURLY!
                    (options {greedy=true;} :"else" LCURLY! (stmt)* RCURLY!)?

{ #ifStmt=#([IFSTMT,"ifStmt"],ifStmt);}
{setLocOnAST((AST)currentAST.root);}
;

// LRM 1.2.3.3      While Loops
whileStmt
  : "while"! LPAREN! expr RPAREN! LCURLY!
          (stmt)*
    RCURLY!
    { #whileStmt=#([WHILESTMT,"whileStmt"],whileStmt);}
{setLocOnAST((AST)currentAST.root);}
  ;
// LRM 1.2.6 Additional functions  (logging)
logStmt :(("LOG_DEBUG" | "LOG_INFO" | "LOG_WARN" | "LOG_ERROR" | "LOG_FATAL")
              LPAREN! expr RPAREN! SEMI!)
{setLocOnAST((AST)currentAST.root);}
              ;

// LRM 1.2.2.3      Function calls  (had to add parameters)
funcCallParams : LPAREN! (expr (COMMA! expr)*)? RPAREN!
{ #funcCallParams = #([FUNCCALLPARAMS, "funcCallParams"], #funcCallParams); }
{setLocOnAST((AST)currentAST.root);}
;
// For standalone statements
funcCallStmt : (ID funcCallParams SEMI!)
{setLocOnAST((AST)currentAST.root);};
// For assignment or other usage that needs typage
funcCallExpr : ID funcCallParams
{ #funcCallExpr = #([FUNCCALLEXPR, "funcCallExpr"], #funcCallExpr); }
{setLocOnAST((AST)currentAST.root);}
;
// Had to add this
// return by itself or with a value.
returnStmt : "return"! (expr)? SEMI!
{ #returnStmt = #([RETURNSTMT, "returnStmt"], #returnStmt); }
{setLocOnAST((AST)currentAST.root);}
;

// LRM 1.2.4 Functions declarations had to change to allow for array and struct
passage
//  structs are referenced as function f( struct sName)
//  arrays are referenced as function f( type[INT]) e.g. function f( int[44] )
```

```
funcDeclParams : LPAREN! ((type_name | ID) ID(LSQR RSQR!)? (COMMA! (type_name |
ID) ID(LSQR RSQR!)?)*)* RPAREN!
{ #funcDeclParams = #([FUNCDECLPARAMS, "funcDeclParams"], #funcDeclParams); }
{setLocOnAST((AST)currentAST.root);}
;
// 1.2.4.1   Local Functions
funcDecl
/* TODO check I think I have to leave the LSQR in so I can tell if this is an
array.*/
  : "function"! ID funcDeclParams "returns"! ("void" | (type_name | ID)(LSQR
RSQR!)?) LCURLY!
       (stmt)*
  RCURLY!
  { #funcDecl = #([FUNCDECL, "funcDecl"], #funcDecl); }
  {setLocOnAST((AST)currentAST.root);}
  ;
// 1.2.4.2   Networked
networkFuncDecl : "network"! funcDecl
{ #networkFuncDecl = #([NETWORKFUNCDECL, "networkFuncDecl"], #networkFuncDecl);
}
{setLocOnAST((AST)currentAST.root);}
;

// LRM 1.2.5 Server
server : "server"! ID LCURLY!
       (
       funcDecl | networkFuncDecl
       /*   Any non declarative statment will be treated as a static block. This
is not in the spec.
             originally I had decls and stmts separate but to allow for struct
instances I combined
             them. What does this mean? This is code ( I guess) that will be
generated into the main.
       */
       |      stmt
       )*
 RCURLY! //EOF!
{
       #server = #([SERVER_AST, "server" ], #server);
       //m_logger.warning("PARSER: server_decls");
}
{setLocOnAST((AST)currentAST.root);}
;

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
//SEMANTICS
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
class NOPELTreeParser extends TreeParser;

options {
    importVocab=NOPELVocab;
    buildAST = false;
}
{
       // Hold on to the current AST so we can provide at least some
       // reasonable error messaging.
       public void holdAST( AST _t )
       {
             NOPELErrorHandler.holdCurrentAST(_t );

       }
       public void doError( String s )
       {
             if (null != s)
                   NOPELErrorHandler.addSemanticError(null,-1,-1,s );
```

```
        }
        public void reportError( String s )
        {
                doError( s );
        }
        public void reportError( RecognitionException re )
        {
                doError( re.getMessage() );
        }
}
id
        :ID {
                        // I dont think I have to do anything here...
                        holdAST( _t);
                }
        ;
////////////////////////////////////////////////////////////////////
//DECLARATIONS
////////////////////////////////////////////////////////////////////
type_name returns [NOPELType t] { t = null; holdAST( _t);} :
        "boolean"       {t = NOPELType.BOOLEAN;}    |
        "date"                  {t = NOPELType.DATE;}           |
        "string"        {t = NOPELType.STRING;}         |
        "int"           {t = NOPELType.INT;}            |
        "real"          {t = NOPELType.REAL;};
primative_decl returns [NOPELDecl d]  {d = null; NOPELType lValueType =
null;holdAST( _t);}// return necessary for structs
:
        // Case 1 No initialization
        #(dec:PRIMATIVE_DECL lValueType = type_name id:ID
        {

                NOPELEnv.getSymbolTable().addPrimative(id.getText(), lValueType);

                d = new NOPELDecl(id.getText(),lValueType);
                NOPELType rValueType = null;
        }
        (ASSIGN! rValueType = exp:expr
                {
                if (null == rValueType)
                        throw new RecognitionException("Could not get type for
expr: "+ exp.getText());

                        if (!NOPELType.isOkToAssign(lValueType,rValueType))
                        {
                                throw new RecognitionException("Cannot assign type:
"+ rValueType.getType()+" to: "+ lValueType.getType());
                        }
                })?
        )
;
struct_decl { NOPELDecl d = null; NOPELType t = null;holdAST( _t);}
:
        #(STRUCT_DECL id:ID
        {
                // Will throw if this is already assigned.
                t = NOPELEnv.getSymbolTable().addType(id.getText());
        }
        #(STRUCTFIELDS
        {
                // After we add the type now enter scope for the properties
                NOPELEnv.enterScope();
        }
        (d = prop:primative_decl
                {
                        // Now add the properties to the type.
                        try {

        ((NOPELStructType)t).addProperty(d.getId(),d.getType());
                        } catch(Throwable te) {
```

```
                        NOPELEnv.leaveScope(); // Make sure the leave scope
on any error...
                        throw new RecognitionException(te.getMessage());

                    }
                }
        )*
        {NOPELEnv.leaveScope();}
        )
        )
        ;
array_decl returns [NOPELType t]{ t= null;NOPELType lValueType =null;holdAST(
_t);}
:
        #(ARRAY_DECL
        (
                lValueType = type_name     // primative case
                |
                structType:ID  // struct case
                {
                        lValueType = NOPELType.getType(structType.getText());
                }
        )
        id:ID
        {
                // Will throw if this is already assigned.
                NOPELEnv.getSymbolTable().addArray(id.getText(), lValueType);
        }
        size:INT
        {
                /** TODO Nothing? I checked size above... I think thats more
semantics and should be here. */
        })
        ;

/* TODO think about making all decls uniform i.e. return the same type. N.B.
   there was the pblm with struct having primatives.*/
decl { NOPELType t = null; NOPELDecl d = null;holdAST( _t);}
        : (d = primative_decl | t = array_decl | struct_decl)
        ;
////////////////////////////////////////////////////////////////////////
//EXPRESSIONS
////////////////////////////////////////////////////////////////////////
booleanRef : "true" | "false";

varRef returns [NOPELType t] { t = null; NOPELType eType=null;holdAST( _t);}
:
                id:ID
                {
                        // throws RE if not defined.
                        t =
NOPELEnv.getSymbolTable().getReturnTypeForId(id.getText());
                }
                /* TODO Check if LSQR and RSQR are just punc & should be out by
now. Without it though,
                    we currently get nondeterminism.
                */
                (LSQR eType = expr
                {
                        // throws exception if invalid.
                        NOPELType.isOkForArrayIndex(id.getText(),eType);
                } RSQR
                ( DOT structProp:ID
                        // We are looking at an array of structs and a property in
the struct.
                        {t =
NOPELType.getTypeForStructRef(t,id.getText(),structProp.getText());}

                 )?
                 |
                 (DOT
                 (structProp1:ID
```

```
                    {t =
NOPELType.getTypeForStructRef(t,id.getText(),structProp1.getText());}
                |
                 "length"
                {
                     // This is a reference to the array length field.
                     t = NOPELType.INT;
                }
                 )) )?
               ;

constRef returns [NOPELType t]
       {
              t = null;
              NOPELType eType = null;
              holdAST( _t);
       }
       :
              #(NEGREALREF {t =  NOPELType.INT;})            |
              #(NEGINTREF  {t =  NOPELType.REAL;})    |
              INT          {t =  NOPELType.INT;}             |
              REAL         {t =  NOPELType.REAL;}            |
              STRING              {t =  NOPELType.STRING;}   |
              DATE         {t =  NOPELType.DATE;}            |
              booleanRef   {t =  NOPELType.BOOLEAN;}
              ;

funcCallExpr returns [NOPELType t] { t= null;holdAST( _t);} :
       #(FUNCCALLEXPR t = funcCallStmt  )
;
basicExpr returns [NOPELType t] { t = null;holdAST( _t);}
:
       t = constRef | t = varRef | t= funcCallExpr
;

relationalOps returns [String s] { s= "Whatever";holdAST( _t);}
: (    GE     {s=">=";}|
       LE     {s="<=";}|
       GT     {s=">";}|
       LT     {s="<";}|
       EQ     {s="=";}|
       NEQ {s="!=";})
;
expr returns [NOPELType t]
       { t = null;
      NOPELType lValueType,rValueType;
      String s = null;
        holdAST( _t);
       }
       :
              // MATH
              #(PLUS lValueType =expr rValueType = expr
              { t = NOPELType.getTypeWhenAddOrSub(lValueType,rValueType);})
       |
              #(MINUS lValueType =expr rValueType = expr
              { t = NOPELType.getTypeWhenAddOrSub(lValueType,rValueType);})
       |
              #(MULT lValueType =expr rValueType = expr
              {t = NOPELType.getTypeWhenMultiOrDiv(lValueType,rValueType);})
       |
              #(DIV lValueType =expr rValueType = expr
              {t = NOPELType.getTypeWhenMultiOrDiv(lValueType,rValueType);})
       |
              /* TODO does spec say anything about relational operators and
type?*/
              /* TODO this is kinda what I want to do but cant get this.
Revisit.
              (s = op:relationalOps lValueType =expr rValueType = expr
              {if (NOPELType.isOkForRlnOp(s,lValueType,rValueType))t =
NOPELType.BOOLEAN;})              |
              */
              // RELATIONAL
```

```
                #(GE lValueType =expr rValueType = expr
                {if (NOPELType.isOkForRlnOp(">=",lValueType,rValueType))t =
NOPELType.BOOLEAN;})
                #(LE lValueType =expr rValueType = expr
                {if (NOPELType.isOkForRlnOp("<=",lValueType,rValueType))t =
NOPELType.BOOLEAN;})
                #(GT lValueType =expr rValueType = expr
                {if (NOPELType.isOkForRlnOp(">",lValueType,rValueType))t =
NOPELType.BOOLEAN;})
                #(LT lValueType =expr rValueType = expr
                {if (NOPELType.isOkForRlnOp("<",lValueType,rValueType))t =
NOPELType.BOOLEAN;})
                #(EQ lValueType =expr rValueType = expr
                {if (NOPELType.isOkForRlnOp("=",lValueType,rValueType))t =
NOPELType.BOOLEAN;})
                #(NEQ lValueType =expr rValueType = expr
                {if (NOPELType.isOkForRlnOp("!=",lValueType,rValueType))t =
NOPELType.BOOLEAN;})
                // NOT
                #(NOT lValueType = expr
                {if (NOPELType.BOOLEAN == lValueType)
                 {
                       t = NOPELType.BOOLEAN;
                 } else {
                       throw new RecognitionException("Cannot negate expression
with type: "+ lValueType);
                 }
                })
                // AND / OR
                #(AND lValueType =expr rValueType = expr
                {if (NOPELType.isOkForAndOrOp("&",lValueType,rValueType))t =
NOPELType.BOOLEAN;})
                #(OR lValueType =expr rValueType = expr
                {if (NOPELType.isOkForAndOrOp("|",lValueType,rValueType))t =
NOPELType.BOOLEAN;})
                // PRIMATIVE
                rValueType = basicExpr
                {t = rValueType;}
        ;
////////////////////////////////////////////////////////////////////////
//STATEMENTS
////////////////////////////////////////////////////////////////////////
assignStmt
:
        {
       NOPELType lValueType,rValueType;
       holdAST( _t);
        }
        #(ASSIGN lValueType = varRef rValueType = expr)
        {
               if (!NOPELType.isOkToAssign(lValueType,rValueType))
               {
                      throw new RecognitionException("Cannot assign type: "+
rValueType.getType()+" to: "+ lValueType.getType());
               }
        }
;

ifStmt : {NOPELType eType = null;holdAST( _t);       }
        #(IFSTMT
        "if" eType = expr
        {
               String ignore = null; // to get rid of warnings on stmts
               // keep track of if and elses returning. If they both do then
               // we need to indicate that in the env so our parent scope
               // knows nothing should continue after this stmt.
               boolean doesElseReturn = false;
               boolean doesIfReturn = false;
               try
               {
                      // Make sure expr is a boolean
                      if (NOPELType.BOOLEAN != eType)
```

```
                        {
                                throw new InvalidExpressionException("cond expr in if
must be boolean. It was: "+ eType.getType());
                        }
                } catch(RecognitionException t)
                {
                        throw t;
                } catch(Throwable t)
                {
                        // We need to be very careful so we always close the scope.
So catch all
                        throw new RecognitionException(t.getMessage());
                 }
        }
    ({NOPELEnv.enterScope();}(ignore =stmt
        {
                        // check if we hit a return statement. If so there should
be no more stmts. Decls ok.
                        if (!NOPELEnv.getSymbolTable().shouldMoreStmtsFollow())
                        {
                                throw new RecognitionException("Unreachable code in
if after return stmt.");
                        }
                        // After each statement check and see if a return was
encountered and if we should continue.
                        if (NOPELEnv.getSymbolTable().wasReturnEncountered())
                        {
                                // indicate no more statements should follow.

       NOPELEnv.getSymbolTable().setShouldMoreStmtsFollow(false);
                                doesIfReturn = true;
                        }
        }
    )*
    {NOPELEnv.leaveScope();})
        ("else"
                ({NOPELEnv.enterScope();}ignore = stmt
        {
                        // check if we hit a return statement. If so there should
be no more stmts. Decls ok.
                        if (!NOPELEnv.getSymbolTable().shouldMoreStmtsFollow())
                        {
                                throw new RecognitionException("Unreachable code in
if after return stmt.");
                        }
                        // After each statement check and see if a return was
encountered and if we should continue.
                        if (NOPELEnv.getSymbolTable().wasReturnEncountered())
                        {
                                // indicate no more statements should follow.

       NOPELEnv.getSymbolTable().setShouldMoreStmtsFollow(false);
                                doesElseReturn = true;
                        }
        }
        )*{NOPELEnv.leaveScope();})?
        {
                if (doesElseReturn && doesIfReturn)
                {
                        // both cases return so we should not allow any stmts after
this.
                        NOPELEnv.getSymbolTable().setReturnEncountered(true);

                //      NOPELEnv.getSymbolTable().setShouldMoreStmtsFollow(false);

                } // I dont think there is an else.
        }
        )
;
        /** TODO Chq where there is a def infinite loop while(true) not on a
variable and no break.
            Java bitches about this and will not compile. */
```

```
whileStmt: {NOPELType eType = null;holdAST( _t); String ignore = null;       }
       #(WHILESTMT
       eType =expr
       {
              NOPELEnv.enterLoopScope();
              try
              {
                     // Make sure expr is a boolean
                     if (NOPELType.BOOLEAN != eType)
                     {
                            throw new InvalidExpressionException("cond expr in
while must be boolean. It was: "+ eType.getType());
                     }
              } catch(RecognitionException t)
              {
                     throw t;
              } catch(Throwable t)
              {
                     // We need to be very careful so we always close the scope.
So catch all
                     throw new RecognitionException(t.getMessage());
               }
              finally {
                     NOPELEnv.leaveScope();
              }
       }
       (ignore = stmt)*
  );
breakOrContinue
       : {holdAST( _t);       }
       id:("break" | "continue") SEMI!
       {
              if (!NOPELEnv.isOkForBreakOrContinue())
              {
                     throw new RecognitionException(id.getText()+" must be
inside a loop.");
              }
       }
       ;
returnStmt : { NOPELType eType = NOPELType.VOID;holdAST( _t);       }
       #(RETURNSTMT
       (eType = expr)?
       {
              NOPELEnv.isReturnTypeCool(eType); // throws if not cool
              // indicate that we found a return stmt.
              NOPELEnv.getSymbolTable().setReturnEncountered(true);
       }
       )
;
structInstantiation : {holdAST( _t);       }structType:ID id:ID
{
       // Make sure what we think is a struct is a struct.
       NOPELType t = NOPELType.getType(structType.getText());
       if (null == t || (!(t instanceof NOPELStructType)))
       {
              throw new RecognitionException("There is no struct of type: "+
structType.getText());
       }
       // we have a valid struct make sure the id is not already defined.
       NOPELEnv.getSymbolTable().addStructVariable(id.getText(),t);
}
;
/* TODO the method isOkForLogStmt is a bit funky in that it will allow any
primative type.
   Spec says only string, but really, I can cast any primative to a string.
I'll
   see when I am generating code if it will be helpful to do arrays or structs
in here too.

*/
logStmt : {NOPELType eType = null;holdAST( _t);       }
("LOG_DEBUG" | "LOG_INFO" | "LOG_WARN" | "LOG_ERROR" | "LOG_FATAL")
```

```
	eType = expr {NOPELType.isOkForLogStmt(eType);}
	;
////////////////////////////////////////////////////////////////////////
//Function Decls Now
////////////////////////////////////////////////////////////////////////
/**
 TODO consider given this impl there is no function overloading. Was this in
the spec?
*/
funcDecl returns [String functionId] {functionId = null;holdAST( _t);	}
:
	#(FUNCDECL
		{
			String ignore = null;
			NOPELType returnType = null;
			// Hold on to the params and we'll add them when we add the
function to the Symbol table
			// This is an (ordered) list of param types. Thats all we
really care about.
			// during walking of this structure though, we will add
params to symbol table.
			// Order is not guaranteed with HashSet or HashTable so I
cant use them
			NOPELType paramType = null;
		}
		funcId:ID
		{

	NOPELEnv.getSymbolTable().addFunction(funcId.getText(),NOPELType.VOID);
			functionId = funcId.getText();
		}
		#(FUNCDECLPARAMS
			{   // Enter a new scope for dealing with parameters
				NOPELEnv.enterFunctionScope();
			}
			(
				(
					paramType = type_name // this is the case for
primatives
					|
					structType:ID  // this is the case for
structs.
					{	// Make sure the struct is defined.
throws if it is not.
						paramType =
NOPELType.getStructType(structType.getText());
					}
				)
				paramName:ID {boolean paramIsArray = false;}
				(LSQR {paramIsArray = true;}
				)?

	{NOPELEnv.getSymbolTable().addFunctionParam(funcId.getText(),paramName.ge
tText(),paramType,paramIsArray);}
			)*
		)
		// At this point we have all the functions parameters. Now we need
the return type.
		//"returns"! ("void" | (type_name | ID)(LSQR RSQR!)?) LCURLY!
		("void"
			{
				returnType = NOPELType.VOID;
			}
			|
			returnType = type_name  // when the return type is a
primative
			|
			structTypeR:ID
			{	// Make sure the struct is defined. throws if it is
not.
				returnType =
NOPELType.getStructType(structTypeR.getText());
```

```
                        }
                )
                {boolean returnTypeIsArray = false;}
                ( LSQR{returnTypeIsArray = true;})?
                {
                        // Now set the return type for this function.

        NOPELEnv.getSymbolTable().setFunctionReturnType(funcId.getText(),returnTy
pe,returnTypeIsArray);
                }
                (ignore = stmt
                {
                        if (!NOPELEnv.getSymbolTable().shouldMoreStmtsFollow())
                        {
                                throw new RecognitionException("Unreachable code
after return stmt in function: "+ funcId.getText());
                        }
                        // After each statement check and see if a return was
encountered and if we should continue.
                        if (NOPELEnv.getSymbolTable().wasReturnEncountered())
                        {
                                // indicate no more statements should follow.

        NOPELEnv.getSymbolTable().setShouldMoreStmtsFollow(false);
                        }
                }
                )*
                {
                        // If the return type was not void make sure we have
encountered a return
                        if( NOPELType.VOID != returnType &&
!NOPELEnv.getSymbolTable().wasReturnEncountered())
                        {
                                throw new RecognitionException("Function:
"+funcId.getText()+". Should return a value of type: "+ returnType.getType()+"
but does not.");
                        }
                        // Leave this scope.
                        NOPELEnv.leaveScope();
                }
        )
  ;
networkFuncDecl : {String functionId = null;holdAST( _t);       }
        #(NETWORKFUNCDECL functionId = funcDecl
        {
                // Mark this function as a network function
                NOPELEnv.getSymbolTable().setAsNetworkFunction(functionId);
        }
        )
;
////////////////////////////////////////////////////////////////////
//Function Calls Now
////////////////////////////////////////////////////////////////////
// return a type so I can reuse this for function call exprs.
funcCallStmt returns [NOPELType t] { t= null;holdAST( _t);       } :
        functionName:ID
        {
                // See if we are being called from inside a function. If so, if
this function
                // is a network function we are in trouble.
                if ( NOPELEnv.isInsideAFunction() &&
NOPELEnv.getSymbolTable().isNetworkFunction(functionName.getText()))
                {
                        throw new RecognitionException("Cannot call the network
function: "+functionName.getText()+". These are not available for usage.");
                }
                // Make sure this is a valid function
                NOPELEnv.getSymbolTable().isFunction(functionName.getText());
                t =
NOPELEnv.getSymbolTable().getReturnTypeForId(functionName.getText());

        }
```

```
        #(FUNCCALLPARAMS
        {
                // Make sure the parameters are cool
                int onParamCount = 0;
                int expectedParamCount =
NOPELEnv.getSymbolTable().getFunctionParamCount(functionName.getText());
                NOPELType paramType = null;
                NOPELType expectedType = null;
        }
        (paramType = expr
        {
                expectedType =
NOPELEnv.getSymbolTable().getFunctionParamType(functionName.getText(),onParamCo
unt++);
                if(!NOPELType.isOkToAssign(expectedType,paramType))
                {
                        throw new RecognitionException("Param "+(onParamCount-1) +"
for func: "+ functionName.getText()+" must have type:
"+expectedType.getType()+" not: "+ paramType.getType());
                }
        }
                (paramType = expr
                {
                        expectedType =
NOPELEnv.getSymbolTable().getFunctionParamType(functionName.getText(),onParamCo
unt++);
                        if(!NOPELType.isOkToAssign(expectedType,paramType))
                        {
                                throw new RecognitionException("Param
"+(onParamCount-1) +" for func: "+ functionName.getText()+" must have type:
"+expectedType.getType()+" not: "+ paramType.getType());
                        }
                }
                )*
        )?
        {
                if (expectedParamCount != onParamCount)
                {
                        throw new RecognitionException("Function: "+
functionName.getText() +" expected: "+ expectedParamCount+" but there were: "+
onParamCount);
                }
        }
        )
;

stmt returns [String type] {type = null;holdAST( _t);      }
/* TODO check situation with the SEMI. This should NOT be here...
                without it though the main will bail when it sees it
*/
        :
        decl                            {type = "Declaration";} |
        assignStmt                      {type = "Assignment";} |
        ifStmt                                  {type = "If";} |
        whileStmt                       {type = "While";} |
        logStmt                         {type = "Log";} |
        breakOrContinue      {type = "break/continue";} |
        structInstantiation {type = "struct";} |
        returnStmt                      {type = "return";} |
        SEMI!                           {type = "semi";}
        ;
server
        :       #(SERVER_AST ID
                        {
                                holdAST( _t);
                                NOPELEnv.enterScope(); NOPELType t = null; String
ignore = null; // to get rid of warnings
                                String stmtType = null;
                        }
                        (ignore = funcDecl | networkFuncDecl |
                        stmtType = stmt
```

```
                        {
                        /* TODO FIX HACK TO ALL SEMIs in MAIN. I should be ashamed
of myself. */
                        if (stmtType != "semi")
                        {
                                        // check if we hit a return statement. If so
there should be no more stmts. Decls ok.
                                        if
(!NOPELEnv.getSymbolTable().shouldMoreStmtsFollow())
                                        {
                                                throw new
RecognitionException("Unreachable code in main after return stmt.Check stmt: "+
stmtType);
                                        }
                                        // After each statement check and see if a
return was encountered and if we should continue.
                                        if
(NOPELEnv.getSymbolTable().wasReturnEncountered())
                                        {
                                                // indicate no more statements should
follow.
        NOPELEnv.getSymbolTable().setShouldMoreStmtsFollow(false);
                                        }

                                }
                        }
                        )*
                        { NOPELEnv.leaveScope(); }
                )
        ;
```

Appendix B NOPELCodeGenerator.g

```
header {

        package com.school.nopel;
        import com.school.nopel.semantics.*;
        import com.school.nopel.codegen.*;
        import com.school.nopel.codegen.decls.*;
                import com.school.nopel.codegen.stmts.*;
}
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
//CODE GENERATION!!!!
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
class NOPELCodeGenerator extends TreeParser;

options {
    importVocab=NOPELVocab;
    buildAST = false;
}
// DONE
id:ID
        ;
////////////////////////////////////////////////////////////////////////////
//DECLARATIONS
////////////////////////////////////////////////////////////////////////////
// DONE
type_name returns [NOPELType t] { t = null;} :
        "boolean"       {t = NOPELType.BOOLEAN;}    |
        "date"                  {t = NOPELType.DATE;}           |
        "string"        {t = NOPELType.STRING;}     |
        "int"           {t= NOPELType.INT;}         |
        "real"          {t = NOPELType.REAL;};
// DONE
```

```
primative_decl returns [PrimativeDecl pd]  {pd = new PrimativeDecl();NOPELType
t= null;} // return structure for struct decl
:
	// Case 1 No initialization
	#(dec:PRIMATIVE_DECL t = type_name id:ID
	{
		pd.setType(t);
		pd.setId(id.getText());
	}
	(ASSIGN! {String exprString = "";} exprString = exp:expr
		{
			pd.setAssignedValue(exprString);
		})?
	)
;
// DONE
struct_decl returns [StructDecl sd]  {sd = new StructDecl();}
:
	#(STRUCT_DECL id:ID
		{ sd.setId(id.getText());}
		#(STRUCTFIELDS {PrimativeDecl prop = null; }
		(prop = primative_decl {sd.addProperty(prop);})*)
	)
	;
/** MAY have some difficulty or wierdness later since I must have defined sizes
for arrays.*/
// DONE
array_decl returns [ArrayDecl ad] {ad = new ArrayDecl();NOPELType t= null;}
:
	#(ARRAY_DECL
	(t = type_name	// primative case
		{ad.setType(t);}
	|
	structName:ID  // struct case
		// I need to create a new NOPELType since I only
		// have a record of declared types during semantic check
		// TODO consider incorp. NOPELEnv into code gen.
		{ad.setType(new NOPELType(structName.getText()));}
	)
	id:ID { ad.setId(id.getText());}
	size:INT {ad.setSize(Integer.parseInt(size.getText()));}
	)
	;
// DONE
decl  returns [TypeDecl t]{ t = null;}
	: (
		t = primative_decl |
		t = array_decl |
		t = struct_decl
		)
	;
////////////////////////////////////////////////////////////////////////
//EXPRESSIONS
////////////////////////////////////////////////////////////////////////

// DONE
booleanRef returns [String s] { s= null;} :
	"true" {s= "true";} |
	"false" {s= "false";}
	;
// DONE
varRef returns [String s] { s = ""; String eString=null;}
:
		id:ID {s += id.getText();}
		(LSQR {s +="[ ";}
		 eString = expr { s+= eString;}
		 RSQR {s+=" ]";}
		 ( DOT {s+=".";}structProp:ID {s+=structProp.getText();})?
		 |
		 (DOT {s+=".";}
			(structProp1:ID {s+=structProp1.getText();}
			|
```

```
                    "length" {s+="length";}
                    )
             )
            )?
            ;
// DONE
constRef returns [String s]
       {
            s = "";
            NOPELType eType = null;
       }
       :
            #(NEGREALREF sr:REAL {s =  "-"+ sr.getText();})          |
            #(NEGINTREF  si:INT  {s =  "-" + si.getText();})         |

            i:INT        { s= i.getText();}          |
            r:REAL       { s= r.getText();}          |
            st:STRING    { s= st.getText(); }        |
            d:DATE              { s= d.getText(); }      |
            s = booleanRef      // booleanRef returns the literal true or
false.
            ;

funcCallExpr returns [String s] { s= null;} :
       #(FUNCCALLEXPR s = funcCallStmt  )
;
// DONE
basicExpr returns [String s] { s = null;}
:
       s = constRef | s = varRef | s= funcCallExpr
;
// DONE
relationalOps returns [String s] { s= "Whatever";}
: (    GE     {s=">=";}|
       LE     {s="<=";}|
       GT     {s=">";}|
       LT     {s="<";}|
       EQ     {s="==";}|
       NEQ {s="!=";})
;
expr returns [String s]
       {
         String sLeft = null;
         String sRight = null;
      s = "";
       }
       :
            // MATH
            #(PLUS sLeft =expr sRight = expr
            { s += (sLeft+"+"+sRight);})          |
            #(MINUS sLeft =expr sRight = expr
            { s += (sLeft+"-"+sRight);})          |
            #(MULT sLeft =expr sRight = expr
            { s += (sLeft+"*"+sRight);})          |
            #(DIV sLeft =expr sRight = expr
            { s += (sLeft+"/"+sRight);})          |
            // RELATIONAL
            #(GE sLeft =expr sRight = expr
            { s += (sLeft+">="+sRight);})         |
            #(LE sLeft =expr sRight = expr
            { s += (sLeft+"<="+sRight);})         |

            #(GT sLeft =expr sRight = expr
            { s += (sLeft+">"+sRight);})          |
            #(LT sLeft =expr sRight = expr
            { s += (sLeft+"<"+sRight);})          |
            #(EQ sLeft =expr sRight = expr
            { s += (sLeft+"=="+sRight);})         |
            #(NEQ sLeft =expr sRight = expr
            { s += (sLeft+"!="+sRight);})         |
            // NOT
            #(NOT sLeft = expr
```

```
                {s+=("!("+ sLeft +")");})         |
                // AND / OR
                #(AND sLeft =expr sRight = expr
                {s+=("("+sLeft +"&"+ sRight+")");})            |
                #(OR sLeft =expr sRight = expr
                {s+=("("+sLeft +"|"+ sRight+")");})            |
                // PRIMATIVE
                s = basicExpr
        ;
////////////////////////////////////////////////////////////////////////
//STATEMENTS
////////////////////////////////////////////////////////////////////////
// DONE
assignStmt returns [String s] { s = "";}
: {String sLeft,sRight;}
        #(ASSIGN sLeft = varRef sRight = expr)
        {s += (sLeft +"="+ sRight+";\n");}
;
// DONE
ifStmt returns [String s] {s= "";}:
        {
                String sExpr = null;
                StatementWrapper sw =null;
                String indent = null;
                int tabCount = 0;
        }
        #(IFSTMT
        "if" sExpr = expr {                                             indent =
ServerDecl.getInstance().getTabs();

                                                        s +=("if ( " + sExpr +"
)\n"+indent+"{\n");
                                                }
        ( {indent = ServerDecl.getInstance().changeTabs(true);}
                (sw =stmt { s +=(indent + sw.toString()+"\n");} )*
        )
        { indent = ServerDecl.getInstance().changeTabs(false); }
        ("else"
        {
                        s +=(indent+ "} else\n"+indent+"{\n");
        }
                        (
                                {indent = ServerDecl.getInstance().changeTabs(true);}

                                sw = stmt {s += (indent + sw.toString()+"\n");}
                        )*
                        {indent = ServerDecl.getInstance().changeTabs(false);}

                        )?
        {
                s +=(indent +"}\n");
        }
        )
;
// DONE
whileStmt returns [String s] {s= "";} :
        {
                String sExpr = null;
                StatementWrapper sw;
                String indent = null;}
    #(WHILESTMT
        sExpr =expr
        {
                indent = ServerDecl.getInstance().getTabs();
                s += ("while ( " + sExpr+" )\n"+indent+"{\n");
        }
                (
                        {indent = ServerDecl.getInstance().changeTabs(true);}
                        sw = stmt
                        {
                                s += (indent + sw.toString()+"\n");
```

```
                                indent = ServerDecl.getInstance().changeTabs(false);
                        }
                )*
        { s+=(indent+"}\n");}
        )
  ;
// DONE
breakOrContinue  returns [String s] {s= "";}
        : ("break" {s="break;\n";}| "continue"{s="continue;\n";})
        ;
/* ARG NOT DONE YET*/
returnStmt  returns [String s] {s= "return "; } : {String sExpr = null;}
        #(RETURNSTMT
        (sExpr = expr
        {
                s += (sExpr);
        }
        )?
        {s+=";\n";}
        )
;
// DONE
structInstantiation returns [String s] {s= "";}
:  structType:ID id:ID
{
        // person dan= (new dans()).new person();
        s = (structType.getText() + " "+ id.getText() + "= ( new "
+ServerDecl.getServerName()+"()).new " + structType.getText() + "();\n");
};
// DONE
logStmt returns [String s] {s= ""; String sExpr = null;}
:
        (       "LOG_DEBUG" {s+= "m_logger.log(java.util.logging.Level.FINER,";}|
                "LOG_INFO"    {s+= "m_logger.log(java.util.logging.Level.INFO,";}|
                "LOG_WARN"    {s+=
"m_logger.log(java.util.logging.Level.WARNING,";}|
                "LOG_ERROR" {s+= "m_logger.log(java.util.logging.Level.SEVERE,";}|
                "LOG_FATAL" {s+= "m_logger.log(java.util.logging.Level.SEVERE,";})

                sExpr = expr {   s+= ("\"\"+"+sExpr + ");\n");}
        ;

////////////////////////////////////////////////////////////////////////
//Function Decls Now
////////////////////////////////////////////////////////////////////////
/**
 TODO consider given this impl there is no function overloading. Was this in
the spec?
*/
funcDecl returns [FunctionDecl functionDecl]
        {
                functionDecl = new FunctionDecl();
                NOPELType paramType;
                NOPELType returnType;
                StatementWrapper sw;
        }
:
        // CONSIDER how I handle client code.
        #(FUNCDECL
           funcId:ID { functionDecl.setId(funcId.getText()); }
               #(FUNCDECLPARAMS
                  (
                                (
                                        paramType = type_name // this is the case for
primatives
                                        |
                                        structType:ID  // this is the case for
structs.
                                        { paramType = new
NOPELType(structType.getText());}
                                )
                                paramName:ID {boolean paramIsArray = false;}
```

```
                        (LSQR {paramIsArray = true;}
                        )?

        {functionDecl.addParameter(paramType,paramName.getText(),paramIsArray);}
                        )*
                )
                ("void" {returnType = NOPELType.VOID;}
                        |
                        returnType = type_name  // when the return type is a
primative
                        |
                        structTypeR:ID {returnType = new
NOPELType(structTypeR.getText());}
                )
                {boolean returnTypeIsArray = false;}
                ( LSQR{returnTypeIsArray = true;})?
                {functionDecl.setReturnType(returnType,returnTypeIsArray);}
                (
                {ServerDecl.getInstance().changeTabs(true);}
                sw = stmt
                {
                        functionDecl.addStatementString(sw.getData());
                }
                {ServerDecl.getInstance().changeTabs(false);}
                )*
        )
 ;
networkFuncDecl returns [FunctionDecl functionDecl] {functionDecl = null;}
:
        #(NETWORKFUNCDECL functionDecl = funcDecl
                // register this with the server so we know to handle it.

        )
;
////////////////////////////////////////////////////////////////////////
//Function Calls Now
////////////////////////////////////////////////////////////////////////
// return a type so I can reuse this for function call exprs.
funcCallStmt returns [String s] { s= ""; String paramExprString = null; boolean
firstParam = true;} :
        functionName:ID { s += (functionName.getText() +"(");}

        #(FUNCCALLPARAMS
        (paramExprString = expr
        {
                if (!firstParam) { s+=", ";} // delimit the params
                s +=(" " + paramExprString); // add the param
                firstParam = false;
        }
                (paramExprString = expr  // mes forgets what this second case is
for??
                {
                        if (!firstParam) { s+=", ";} // delimit the params
                        s +=(" " + paramExprString); // add the param
                        firstParam = false;
                }
                )*
        )?
        )
;

stmt returns [StatementWrapper sw] {sw = null; TypeDecl td = null; String
s=null;}
/* TODO  Can I collapse all these case? */
        :
        td = decl                               { sw = new DeclWrapper(td);}
        |
        s = assignStmt              {sw = new StatementWrapper(s);}         |
        s = ifStmt                      {sw = new StatementWrapper(s);}
        |
        s = whileStmt                   {sw = new StatementWrapper(s);}
        |
```

```
        s = logStmt                             {sw = new StatementWrapper(s);}
        |
        s = breakOrContinue             {sw = new StatementWrapper(s);}         |
        s = structInstantiation         {sw = new StatementWrapper(s);}         |
        s = returnStmt                          {sw = new StatementWrapper(s);}
        |
        SEMI                                    {sw = new StatementWrapper("//ROGUE
SEMI FIX ME. SOME DECL DID NOT EAT THIS\n");}
        ;
// Print out the server/client file at the end.
server returns [String s] { s=null; StatementWrapper sw = null;}
        :       #(SERVER_AST name:ID
                {
                        String sString = null;
                        FunctionDecl fd= null;
                        TypeDecl td;
                        String indent = "";

                        ServerDecl sd = ServerDecl.getInstance();
                        //sd.setPackageToWriteTo("com.nopel.autogen."+
name.getText());
                        sd.setServerName(name.getText());
                        sd.start();
                }
                (
                                (
                                        {sd.changeTabs(true);}
                                        sw = stmt {sd.addStatment(sw);
sd.changeTabs(false);}
                                        |
                                        fd = funcDecl
                                        {
                                                sd.addStatmentBlock(); // writes out
static block for static stmts
                                                sd.addFunction( fd );

                                        }
                                        |
                                        fd = networkFuncDecl
                                        {
                                                sd.addStatmentBlock(); // writes out
static block for static stmts
                                                sd.addNetworkFunction( fd );

                                        }
                                )*
                                // writes out static block for static stmts
                                {sd.addStatmentBlock();}
                )
                {
                        sd.finish();
                        s = sd.getCode("");

                }
                )
        ;
```

Appendix C NOPELCommonAST.java

```
package com.school.nopel;

import antlr.CommonAST;
/**
 * Basically acts like a C# delgator without Line # and Col #
 *
 * Used in the Parser/Semantics so I can keep track of
 * roughly where this node is located textually in the
 * program. Each Parser returned AT will return this
 * type.
```

```
 *
 * I will replace the returned AST with this instance
 * on which I will have the line number and column
 * number set.
 */
public class NOPELCommonAST extends CommonAST {

	static final long serialVersionUID = 7221976; // For comp warning
	private int m_lineNum = -1;
	private int m_colNum = -1;

	public NOPELCommonAST()
	{
		super();
	}
	public int getColumn() {
		return m_colNum;
	}
	public int getLine() {
		return m_lineNum;
	}
	public void setColumn(int _col) {
		m_colNum = _col;
	}
	public void setLine(int _line) {
		m_lineNum = _line;
	}
}
```

Appendix C NOPELASTFactory.java

```
package com.school.nopel;

import antlr.ASTFactory;

public class NOPELASTFactory extends ASTFactory
{
	/**
	 * Same logic as base class but return the NOPEL AST node so
	 * I can set the line and column number.
	 */
	public Class getASTNodeType(int tokenType) {
		// try node specific class
		if ( tokenTypeToASTClassMap!=null ) {
			Class c = (Class)tokenTypeToASTClassMap.get(new Integer(tokenType));
			if ( c!=null ) {
				return c;
			}
		}

		// try a global specified class
		if (theASTNodeTypeClass != null) {
			return theASTNodeTypeClass;
		}

		// Use my own so I can set the line and column number.
		return NOPELCommonAST.class;
	}

}
```

All Other files

```
//////////////////////////////////////////
//////////////////////////////////////////
//ClientStubGenerator.java
```

```
//~~//////////////////////////////////////
//////////////////////////////////////
package com.school.nopel.codegen;
import java.util.ArrayList;
import java.util.Iterator;

import com.school.nopel.codegen.FileGenerator;
import com.school.nopel.codegen.decls.CodeGenerator;
import com.school.nopel.codegen.decls.FunctionDecl;
import com.school.nopel.codegen.decls.PrimativeDecl;
import com.school.nopel.codegen.decls.ServerDecl;
import com.school.nopel.codegen.decls.StructDecl;
import com.school.nopel.codegen.decls.FunctionDecl.Parameter;
import com.school.nopel.semantics.NOPELType;
/**
 * This write out the clients for the servers.
 */
public class ClientStubGenerator implements CodeGenerator, FileGenerator
{
/*
	public static boolean m_supressAutoGenMsg = false;
	/**
	 * Network functions to listen to.
	 */
	public ArrayList m_netFunctions = new ArrayList();
	public ArrayList m_structs = new ArrayList();
	protected static String m_packageToWriteTo = "com.nopel.client";
	/**
	 * Name for the server we will be contacting.
	 */
	protected static String m_serverName;
	/**
	 * Buffer for iteratively constructing the client code.
	 */
	protected StringBuffer m_clientCode = new StringBuffer("");
	private static ClientStubGenerator m_instance = null;
	public static ClientStubGenerator getInstance()
	{
		if (null == m_instance) m_instance = new ClientStubGenerator();
		return m_instance;
	}
	public void setServerName(String _serverName) { m_serverName = _serverName;}
	public static String getServerName() { return m_serverName;}
	private ClientStubGenerator()
	{
		super();
	}
	public void setFunctions(ArrayList _functions)
	{
		m_netFunctions = _functions;
	}
	public void setStructs(ArrayList _structs)
	{
		m_structs = _structs;
	}
	/**
	 * Generates the contents for .java file with a main method.
	 */
	public String getCode(String _tabs)
	{
		if (null != m_packageToWriteTo)
		{
			m_clientCode.append("package ");
			m_clientCode.append(m_packageToWriteTo);
			m_clientCode.append(";\n");
		}
		m_clientCode.append(ServerDecl.getAutoGenWarning());
		m_clientCode.append("public class "+m_serverName+"Client\n{\n");
```

```
		m_clientCode.append("\tstatic java.util.logging.Logger m_logger =
java.util.logging.Logger.getLogger("+m_serverName+"Client.class.getName());\n");

		m_clientCode.append("\t/**Initialization for a date decl.*/\n\tprivate static java.util.Date _initDate(String
_date)\n");
		m_clientCode.append("\t{\n\t\ttry {\n\t\t\treturn new
java.text.SimpleDateFormat(\"ddMMMyyyy\").parse(_date);\n\t\t} catch (java.text.ParseException pe){\n\t\t\treturn
null;\n\t\t}\n\t}\n");


		// Write out the structs (inner class) if there are any
		Iterator it = m_structs.iterator();
		while (it.hasNext())
		{
			m_clientCode.append(((StructDecl)it.next()).getCode("\t"));
			m_clientCode.append("\n");
		}
		it = m_netFunctions.iterator();
		FunctionDecl fd = null;
		while (it.hasNext())
		{
			fd = (FunctionDecl) it.next();
			m_clientCode.append(getContentHandler(fd));
		}

		it = m_netFunctions.iterator();
		Iterator paramIt = null;
		while (it.hasNext())
		{
			fd = (FunctionDecl) it.next();
			////////////////////////////////////////////////////
			// Hold the IP/Port
			////////////////////////////////////////////////////
			m_clientCode.append("		protected static int m_"+ fd.getId()+"Port = -1;\n");

			m_clientCode.append("		protected static String m_"+ fd.getId()+"IP = \"\";\n");

			////////////////////////////////////////////////////
			// Set the IP/Port
			////////////////////////////////////////////////////
			m_clientCode.append(
				"		public static void set"+fd.getId()+"Config(String _ip, int _port)\n"
+
				"		{\n" +
				"			m_"+ fd.getId()+"IP = _ip;\n" +

				"			m_"+ fd.getId()+"Port = _port;\n" +

				"		}\n"
			);
			////////////////////////////////////////////////////
			// Actual Function call
			////////////////////////////////////////////////////
			m_clientCode.append("		public static ");
			m_clientCode.append(PrimativeDecl.getJavaTypeString(fd.getReturnType()));
			if (fd.isReturnArray())
			{
				m_clientCode.append("[]");
			}
			m_clientCode.append(" "+ fd.getId()+"(");
			if (fd.getParameters().size() < 1)
			{
				m_clientCode.append(")\n");
			}
			else
			{
				//m_clientCode.append("\n");
				////////////////////////////////////////////////////
				// Take care of the parameters
				////////////////////////////////////////////////////
```

```
                paramIt = fd.getParameters().iterator();
                Parameter p =null;
                while (paramIt.hasNext())
                {
                        p = (Parameter)paramIt.next();
                        m_clientCode.append(PrimativeDecl.getJavaTypeString(p.m_type)+" "+
p.m_id);
                        if (p.m_isArray)
                                m_clientCode.append("[]");

                        if (paramIt.hasNext())
                                m_clientCode.append(",");
                }
                m_clientCode.append(")\n");
        }
        m_clientCode.append("           {\n");
        ////////////////////////////////////////////////
        // Take care of the calling code.
        ////////////////////////////////////////////////
        m_clientCode.append(
                "                       java.net.Socket socket = null;\n"+
                "                       String fromServer;\n"+
                "                       java.io.BufferedReader in = null;\n"+
                "                       "+getInitializedReturnObject(fd)+"\n"+
                "                       try {\n"+
                getParameters(fd) +
                "                               socket = new
java.net.Socket(java.net.InetAddress.getByName(m_"+fd.getId()+"IP),m_"+fd.getId()+"Port);\n"+
                "
    socket.getOutputStream().write(outMsg.getBytes());\n" +
                "                               socket.getOutputStream().flush();\n"+
                "                       in = new java.io.BufferedReader(new
java.io.InputStreamReader(socket.getInputStream()));\n"+
                "                       "+fd.getId()+"ContentHandler ch = (new
"+m_serverName+"Client()).new "+fd.getId()+"ContentHandler();\n"+

                "                       while ((fromServer = in.readLine()) != null)\n"+
                "                       {\n");
        if (NOPELType.BOOLEAN == fd.getReturnType() ||
                NOPELType.DATE == fd.getReturnType() ||
                NOPELType.REAL == fd.getReturnType() ||
                NOPELType.INT == fd.getReturnType() ||
                NOPELType.STRING == fd.getReturnType())
        {
                m_clientCode.append("                   org.xml.sax.XMLReader r =
org.xml.sax.helpers.XMLReaderFactory.createXMLReader();\n"+
                "                       r.setContentHandler(ch);\n"+
                "                       r.parse(new org.xml.sax.InputSource(new
java.io.StringReader(fromServer)));\n"+
                "                               "+getSetReturnObject(fd)+"\n"
                );
        } else {
                // we are a struct
                if (fd.isReturnArray())
                {
                        // TODO yikes I can get into trouble here if someone declares a variable
pWhatever...
                m_clientCode.append(
                "                               String pWhatever[] =
fromServer.split(\"</"+PrimativeDecl.getJavaTypeString(fd.getReturnType())+">\");\n" +
                "                               out = new
"+PrimativeDecl.getJavaTypeString(fd.getReturnType())+"[pWhatever.length-1];\n"+
                "                               for (int i =0; i <
pWhatever.length-1; ++i)\n"+
                "                               {\n" +
                "                                       out[i] = (new
"+m_serverName+"Client()).new "+PrimativeDecl.getJavaTypeString(fd.getReturnType())+"(pWhatever[i]);\n"+
                "                               }\n");
                } else {
                        m_clientCode.append(
```

```
                                                "                                                       out = (new
"+m_serverName+"Client()).new "+PrimativeDecl.getJavaTypeString(fd.getReturnType())+"(fromServer);\n");

                                        }
                                }
                                        m_clientCode.append("                                   }\n"+
                                                "                               } catch(Exception e) {\n"+
                                                "                                       try {socket.close();}catch(Exception t)
{}\n"+
                                                "                               }\n"+
                                                "                               return out;\n"

                                        );
                                        ////////////////////////////////////////////////
                                        m_clientCode.append("           }\n");
                        }
                        m_clientCode.append("}\n");


                        return m_clientCode.toString();
        }
        protected String getParameters(FunctionDecl fd)
        {
                        StringBuffer out = new StringBuffer("");
                        out.append("                                    String outMsg =\"<NOPELMessage>\";\n");
                        Iterator it = fd.getParameters().iterator();
                        Parameter p;
                        while(it.hasNext())
                        {
                                p = (Parameter) it.next();
                                if (p.m_isArray)
                                {
                                        out.append("                                    outMsg +=\"<Array>\";\n");
                                        out.append("                                    for (int i=0; i < "+p.m_id+".length; ++i)\n");
                                        out.append("                                    {\n");
                                        /*
                                        out.append("                                            outMsg
+=\"<"+p.m_id+">\";\n");
                                        out.append("                                            outMsg +="+p.m_id+"[i];\n");

                                        out.append("                                            outMsg
+=\"</"+p.m_id+">\";\n");
                                        */
                                        out.append("                                            outMsg
+=\"<"+PrimativeDecl.getJavaTypeString(p.m_type)+">\";\n");
                                        out.append("                                            outMsg +="+p.m_id+"[i];\n");

                                        out.append("                                            outMsg
+=\"</"+PrimativeDecl.getJavaTypeString(p.m_type)+">\";\n");
                                        out.append("                                    }\n");
                                        out.append("                                    outMsg +=\"</Array>\";\n");
                                } else {
                                        out.append("                                    outMsg +=\"<"+p.m_id+">\";\n");
                                        out.append("                                    outMsg +="+p.m_id+";\n");

                                        out.append("                                    outMsg +=\"</"+p.m_id+">\";\n");

                                }
                        }
                        // This is f'd up. There must be a \n or my server will baulk
                        out.append("                                    outMsg +=\"</NOPELMessage>\\n\";\n");
                        return out.toString();
        }
        protected String getInitializedReturnObject(FunctionDecl _fd)
        {
                        if (_fd.isReturnArray())
                        {
                                return PrimativeDecl.getJavaTypeString(_fd.getReturnType()) +" out[] = null;";
                        }
                        if ( _fd.getReturnType().equals(NOPELType.REAL))
```

```
		{
			return "double out = -1;";
		}
		if ( _fd.getReturnType().equals(NOPELType.INT))
		{
			return "int out = -1;";
		}
		if ( _fd.getReturnType().equals(NOPELType.BOOLEAN))
		{
			return "boolean out = false;";
		}
		return PrimativeDecl.getJavaTypeString(_fd.getReturnType()) +" out = null;";
	}
	protected String getSetReturnObject(FunctionDecl _fd)
	{
		if ( _fd.isReturnArray())
		{

	NOPELCodeGenHelper.parseXMLForArrayVarInitialization(_fd.getReturnType(),m_serverName+"Client","fromServer",
"out");
		}
		if ( _fd.getReturnType().equals(NOPELType.REAL))
		{
			return "out = Double.parseDouble(ch.value);\n";
		}
		if ( _fd.getReturnType().equals(NOPELType.INT))
		{
			return "out = Integer.parseInt(ch.value);\n";
		}
		if ( _fd.getReturnType().equals(NOPELType.BOOLEAN))
		{
			return "out = Boolean.valueOf(ch.value).booleanValue();";
		}
		if ( _fd.getReturnType().equals(NOPELType.DATE))
		{
			return "out = _initDate(ch.value);";
		}
		if ( _fd.getReturnType().equals(NOPELType.STRING))
		{
			return "out = new "+PrimativeDecl.getJavaTypeString(_fd.getReturnType())+"(ch.value);";
		}
		// structs (can parse message without the content handler
		return "out = (new "+m_serverName+"Client()).new
"+PrimativeDecl.getJavaTypeString(_fd.getReturnType())+"(fromServer);";

	}
	public static String getPackageToWriteTo()
	{
		return m_packageToWriteTo;
	}
	public void setPackageToWriteTo(String _pkg)
	{
		m_packageToWriteTo = _pkg;
	}
	public static String getContentHandler(FunctionDecl _fd)
	{
		StringBuffer out = new StringBuffer();
		String _tabs = "\t";
		out.append(_tabs +"class "+_fd.getId()+"ContentHandler implements org.xml.sax.ContentHandler\n");
		out.append(_tabs +"{\n");
		_tabs += "\t";
		out.append(_tabs +"public String value = \"\";\n");
		out.append(_tabs +"boolean go = false;\n");
		out.append(_tabs +"boolean element = false;\n");
		out.append(_tabs +"public "+_fd.getId()+"ContentHandler() {}\n");
		out.append(_tabs +"public void characters(char[] ch, int start, int length) throws
org.xml.sax.SAXException\n");
		out.append(_tabs +"{\n");
		_tabs += "\t";
		out.append(_tabs +"if (go)\n");
```

```
                        out.append(_tabs +"{\n");
                        _tabs += "\t";
                        if (_fd.isReturnArray())
                        {
                                out.append(_tabs +"if (element)\n");
                                out.append(_tabs +"{\n");
                                _tabs += "\t";
                                out.append(_tabs +"value += (new String(ch,start,length)) +\",\";\n");
                                _tabs ="\t\t\t\t";
                                out.append(_tabs +"}\n");
                        } else {
                                out.append(_tabs +"value += new String(ch,start,length);\n");
                        }
                        _tabs = "\t\t\t";
                        out.append(_tabs +"}\n");
                        _tabs = "\t\t";
                        out.append(_tabs +"}\n");
                        _tabs = "\t\t";
                        out.append(_tabs +"public void endElement(String namespaceURI, String localName, String qName) throws
org.xml.sax.SAXException {\n");
                        _tabs += "\t";
                        if (_fd.isReturnArray())
                        {
                                out.append(_tabs +"if (localName.equals(\"Array\"))\n");
                                out.append(_tabs +"go = false;\n");
                        } else {
                                out.append(_tabs +"if
(localName.equals(\""+PrimativeDecl.getJavaTypeString(_fd.getReturnType())+"\"))\n");
                                out.append(_tabs +"element = false;\n");
                        }
                        if (_fd.isReturnArray())
                        {
                                out.append(_tabs +"if
(localName.equals(\""+PrimativeDecl.getJavaTypeString(_fd.getReturnType())+"\"))\n");
                                out.append(_tabs +"element = false;\n");
                        }
                        _tabs = "\t\t";
                        out.append(_tabs +"}\n");
                        out.append(_tabs +"public void startElement(String namespaceURI, String localName, String qName,
org.xml.sax.Attributes atts) throws org.xml.sax.SAXException {\n");
                        _tabs += "\t";
                        if (_fd.isReturnArray())
                        {
                                out.append(_tabs +"if (localName.equals(\"Array\"))\n");
                                out.append(_tabs +"go = true;\n");
                                out.append(_tabs +"if
(localName.equals(\""+PrimativeDecl.getJavaTypeString(_fd.getReturnType())+"\"))\n");
                                out.append(_tabs +"element = true;\n");
                        } else {
                                out.append(_tabs +"if
(localName.equals(\""+PrimativeDecl.getJavaTypeString(_fd.getReturnType())+"\"))\n");
                                _tabs += "\t";
                                out.append(_tabs +"go = true;\n");
                        }
                        _tabs = "\t\t";
                        out.append(_tabs +"}\n");
                        _tabs = "\t\t";
                        out.append(_tabs +"public void endDocument() throws org.xml.sax.SAXException {}\n");

                        out.append(_tabs +"public void endPrefixMapping(String prefix) throws org.xml.sax.SAXException {}\n");
                        out.append(_tabs +"public void ignorableWhitespace(char[] ch, int start, int length) throws
org.xml.sax.SAXException {}\n");
                        out.append(_tabs +"public void processingInstruction(String target, String data) throws
org.xml.sax.SAXException {}\n");
                        out.append(_tabs +"public void setDocumentLocator(org.xml.sax.Locator locator) {}\n");
                        out.append(_tabs +"public void skippedEntity(String name) throws org.xml.sax.SAXException {}\n");
                        out.append(_tabs +"public void startDocument() throws org.xml.sax.SAXException {}\n");
                        out.append(_tabs +"public void startPrefixMapping(String prefix, String uri) throws org.xml.sax.SAXException
{}\n");
                        _tabs = "\t";
```

```
                        out.append(_tabs +"};\n");
                        return out.toString();
        };
}
////////////////////////////////////////
////////////////////////////////////////
//ArrayDecl.java
//~~////////////////////////////////////
////////////////////////////////////////
package com.school.nopel.codegen.decls;

/**
 *
 */
public class ArrayDecl extends TypeDecl
{
        // List of properties on this decl.
        int m_size = 0;
        public ArrayDecl()
        {
                        super();
        }
        public int getSize() {
                        return m_size;
        }
        public void setSize(int m_size) {
                        this.m_size = m_size;
        }
        public String getCode(String _tabs)
        {
                        // TODO add another type method. This is messed up that I have to do this.
                        String type = PrimativeDecl.getJavaTypeString(getType());
                        return (_tabs + type + " " + getId() +"[] = "+"new "+ type +"["+ m_size +"];");
        }
        public String toString()
        {
                        return getCode("");
        }
}
////////////////////////////////////////
////////////////////////////////////////
//CodeGenerator.java
//~~////////////////////////////////////
////////////////////////////////////////
package com.school.nopel.codegen.decls;

public interface CodeGenerator {
        public String getCode(String _tabs);
}
////////////////////////////////////////
////////////////////////////////////////
//Exportable.java
//~~////////////////////////////////////
////////////////////////////////////////
package com.school.nopel.codegen.decls;
/**
 * Interface for something that can be exported from client to server.
 * Basically defines the method to generate the XML representation of the
 * object. Kinda like serialization but ummm, dummer.
 *
 */
public interface Exportable {
        public String getExportableXML();
}
////////////////////////////////////////
////////////////////////////////////////
//FunctionDecl.java
//~~////////////////////////////////////
////////////////////////////////////////
package com.school.nopel.codegen.decls;
```

```
import java.util.ArrayList;
import java.util.Iterator;

import com.school.nopel.semantics.NOPELType;

/**
 *
 */
public class FunctionDecl extends TypeDecl implements CodeGenerator
{
	/**
	 * The tabs to prepend to each stmt.
	 */
	protected String m_tabs = "";
	protected NOPELType m_returnType = null;
	protected ArrayList m_parameters = new ArrayList();
	protected ArrayList m_stmts = new ArrayList();
	protected boolean m_returnArray = false;
	public boolean isReturnArray() { return m_returnArray;}
	/**
	 * The code that we are generating here.
	 */
	protected StringBuffer m_functionCode = new StringBuffer();

	public FunctionDecl(){super();}


	public NOPELType getReturnType() {
		return m_returnType;
	}
	public void setReturnType(NOPELType type,boolean _isArray) {
		m_returnType = type;
		m_returnArray = _isArray;
	}
	public void addParameter(NOPELType _type, String _id,boolean _isArray)
	{
		m_parameters.add(new Parameter(_type,_id,_isArray));
	}
	public ArrayList getParameters()
	{
		return m_parameters;
	}
	public String getCode()
	{
		return getCode("");
	}
	public String getCode(String _tabs)
	{
		start(_tabs);
		Iterator it = m_stmts.iterator();
		while (it.hasNext())
		{
			m_functionCode.append( "\t"+_tabs + it.next().toString() );
		}
		finish();
		return m_functionCode.toString();
	}
	public void addStatementString(Object _stmt)
	{
		m_stmts.add(_stmt);
	}
	/**
	 * Once we have the function declaration we can start writing out code to
	 * our code buffer.
	 */
	public void start(String _tabs)
	{
		m_tabs = _tabs;
		m_functionCode.append(m_tabs+ "protected ");
```

```
                m_functionCode.append(PrimativeDecl.getJavaTypeString(m_returnType));

                if (m_returnArray)
                        m_functionCode.append("[]");

                m_functionCode.append(" ");
                m_functionCode.append(m_id);
                m_functionCode.append("(" );
                Iterator it = m_parameters.iterator();
                int numParams = m_parameters.size();
                Parameter p = null;
                while (it.hasNext())
                {
                        p = (Parameter) it.next();
                        m_functionCode.append( PrimativeDecl.getJavaTypeString(p.m_type) +" " + p.m_id );
                        if (p.m_isArray)
                                m_functionCode.append("[]");

                        if (numParams-- > 1)
                        {
                                m_functionCode.append(", ");
                        }
                }
                m_functionCode.append(" )\n" );
                m_functionCode.append(m_tabs+"{\n" );
        }
        public void finish()
        {
                m_functionCode.append(m_tabs+"}\n");
        }
        public class Parameter
        {
                public NOPELType m_type;
                public String m_id;
                public boolean m_isArray;
                protected Parameter(NOPELType _type, String _id, boolean _isArray)
                {
                        m_type = _type;
                        m_id = _id;
                        m_isArray = _isArray;
                }
        }
}
//////////////////////////////////////////
//////////////////////////////////////////
//PrimativeDecl.java
//~~//////////////////////////////////////
//////////////////////////////////////////
package com.school.nopel.codegen.decls;


import com.school.nopel.semantics.NOPELType;

/**
 * Represents a Primative Declaration. Used to generated code.
 */
public class PrimativeDecl extends TypeDecl implements Exportable
{
                String m_assignedValue;
                public PrimativeDecl()
                {
                        super();
                }
                public String getAssignedValue() {
                        return m_assignedValue;
                }
                public void setAssignedValue(String value) {
                        m_assignedValue = value;
                }
                public String getCode(String _tabs)
```

```
		{
			// Pretty standard except for dates.
			if (NOPELType.DATE == getType())
			{
				return _tabs + getJavaTypeString(getType()) +" "+getId() +" =
_initDate(\""+getAssignedValue()+"\");";
			} else {
				String out = _tabs+getJavaTypeString(getType()) + " " + getId();
				if (null != m_assignedValue)
				{
					out += (" = " + m_assignedValue);
				}
				out +=";";
				return out;
			}
		}
		public String toString()
		{
			return getCode("");
		}
		/**
		 * @TODO REFACTOR AND MOVE ME OUT OF PRIM DECL
		 * @param _nt
		 * @return
		 */
		public static String getJavaTypeString(NOPELType _nt)
		{
			if (NOPELType.BOOLEAN == _nt)
				return "boolean";
			if (NOPELType.DATE == _nt)
				return "java.util.Date";
			if (NOPELType.INT == _nt)
				return "int";
			if (NOPELType.REAL == _nt)
				return "double";
			if (NOPELType.STRING == _nt)
				return "String";
			return _nt.getType();
		}
		public String getExportableXML()
		{
			String out = "";
			out += "<"+m_id+" type=\\\""+getJavaTypeString(m_type)+"\\\">";
			out += ""+m_assignedValue;
			out += "</"+m_id+">";
			return out;
		}
}
////////////////////////////////////////
////////////////////////////////////////
//ServerDecl.java
//~~////////////////////////////////////
////////////////////////////////////////
package com.school.nopel.codegen.decls;

import java.util.ArrayList;
import java.util.Date;
import java.util.Iterator;

import com.school.nopel.codegen.FileGenerator;
import com.school.nopel.codegen.NOPELCodeGenHelper;
import com.school.nopel.codegen.decls.FunctionDecl.Parameter;
import com.school.nopel.codegen.stmts.DeclWrapper;
import com.school.nopel.codegen.stmts.StatementWrapper;
import com.school.nopel.semantics.NOPELType;
/**
 * This write out the declaration for a server
 */
public class ServerDecl implements CodeGenerator, FileGenerator
{
```

```
/**
 * Used when generating main
 */
public ArrayList m_structs = new ArrayList();
/**
 * Used when generating main
 */
public ArrayList m_netFunctions = new ArrayList();
/**
 * Flag so we can do exact match reg tests.
 */
public static boolean m_supressAutoGenMsg = false;
protected static String m_tab = "";
protected int m_numberOfTabs = 0;
/**
 * For pretty files, tabify
 * @param _add
 */
public String changeTabs(boolean _add)
{
		if (_add) {m_numberOfTabs++; } else { m_numberOfTabs--;}
		m_tab = "";
		for (int i=0; i < m_numberOfTabs; ++i) m_tab +="\t";

		return m_tab;
}
public int getTabCount() { return m_numberOfTabs;}
/**
 * When we enter a new scope we need to reset tabs.
 */
public String setTabs(int _count) {
		m_numberOfTabs = _count -1;
		return changeTabs(true);
}
public static String getTabs() { return m_tab;}
protected static String m_packageToWriteTo = "com.nopel.server";
/**
 * Name for the server
 */
protected static String m_serverName;
/**
 * Buffer for iteratively constructing the server code.
 */
protected StringBuffer m_serverCode = new StringBuffer("");
private static ServerDecl m_instance = null;
public static ServerDecl getInstance()
{
		if (null == m_instance) m_instance = new ServerDecl();
		return m_instance;
}
public void setServerName(String _serverName) { m_serverName = _serverName;}
public static String getServerName() { return m_serverName;}
private ServerDecl() { super();}
/**
 * Generates the contents for .java file with a main method.
 */
public void start()
{
		if (null != getPackageToWriteTo())
		{
			m_serverCode.append("package ");
			m_serverCode.append(getPackageToWriteTo());
			m_serverCode.append(";\n");
		}

		m_serverCode.append(getAutoGenWarning());
		// TODO consider extending some base class that does all the real work.
		m_serverCode.append("public class "+m_serverName+"\n{\n");
```

```
		m_serverCode.append("\tstatic java.util.logging.Logger m_logger =
java.util.logging.Logger.getLogger("+m_serverName+".class.getName());");

		m_serverCode.append("\t/**Initialization for a date decl.*/\n\tprivate static java.util.Date _initDate(String
_date)\n");

		m_serverCode.append("\t{\n\t\ttry {\n\t\t\treturn new
java.text.SimpleDateFormat(\"ddMMMyyyy\").parse(_date);\n\t\t} catch (java.text.ParseException pe){\n\t\t\treturn
null;\n\t\t}\n\t}\n");

		changeTabs(true);
	}
	protected ArrayList m_stmtBuffer = new ArrayList();
	public void addStatment(StatementWrapper _sw)
	{
		if (_sw instanceof DeclWrapper)
		{
			Object data = _sw.getData();
			if (data instanceof StructDecl)
			{
				m_serverCode.append(((StructDecl)data).getCode("")+"\n");
				m_structs.add(data);
			} else {
				changeTabs(false); // TODO FIX THIS changeTabs
				m_serverCode.append(m_tab+"private static " +((TypeDecl)data).getCode("")+"\n");
				changeTabs(true);
			}
		} else {
			m_stmtBuffer.add(_sw);
		}
	}
	public void addStatmentBlock()
	{
		if (m_stmtBuffer.size() >0)
		{
			m_serverCode.append(m_tab + "static\n"+m_tab+"{\n");
			Iterator it = m_stmtBuffer.iterator();
			StatementWrapper sw;
			changeTabs(true); // indent in static block
			while (it.hasNext())
			{
				sw = (StatementWrapper) it.next();
				m_serverCode.append(m_tab+ sw.getData());
			}
			changeTabs(false);
			m_serverCode.append(m_tab + "}\n");
			m_stmtBuffer.clear();
		}
	}
	protected String getSectionDelimiter()
	{
		return m_tab+"// ******************************************************\n";
	}
	/**
	 * Puts in the last end brace.
	 */
	public void finish()
	{
		// We need to create the workers for each network function
		Iterator it = m_netFunctions.iterator();
		while (it.hasNext())
		{
			m_serverCode.append(getSectionDelimiter());
			writeNetworkWorker((FunctionDecl) it.next());
		}
		m_serverCode.append(m_tab+"public static void main(String[] args) throws Exception\n"+m_tab+"{\n");
		// Get the ports to use from the command line.
		m_serverCode.append(
		"			java.util.ArrayList ports = new java.util.ArrayList();\n"+
		"			m_logger.log(java.util.logging.Level.INFO,\"Enter the ports to use:\");\n"+
```

```
                "                       java.io.BufferedReader m_reader = new java.io.BufferedReader(new
java.io.InputStreamReader(System.in));\n" +
                "                       int i = 0;\n"
                );
                it = m_netFunctions.iterator();
                FunctionDecl fd = null;
                while (it.hasNext())
                {
                        fd = (FunctionDecl) it.next();
                        m_serverCode.append(
                                        "\n                     System.out.print(\"Enter the port for function
"+fd.m_id+":\");\n"+
                                        "                       ports.add(m_reader.readLine());\n"
                                        );
                }
                // Start the services
                m_serverCode.append(
                "                       // For each networked function kick off a thread for a socket that handles\n" +
                "                       // that call.\n"
                );
                it = m_netFunctions.iterator();
                while (it.hasNext())
                {
                        fd = (FunctionDecl) it.next();
                        m_serverCode.append(
                                        "                       "+fd.m_id+"Runnable "+fd.m_id+"Inst = (new
"+m_serverName+"()).new "+fd.m_id+"Runnable();\n" +
                                        "                       java.lang.Thread "+fd.m_id+"Thread = new
java.lang.Thread("+fd.m_id+"Inst);\n" +
                                        "\n                     "+fd.m_id+"Inst.m_port =
Integer.parseInt(ports.get(i++).toString());\n"+
                                        "                       "+fd.m_id+"Thread.start();\n"

                                        );
                }
                m_serverCode.append(
                                        "                       // If users presses a key kill all the servers\n");
                m_serverCode.append(
                                        "\n                     m_logger.log(java.util.logging.Level.INFO,\"Hit any char to kill
all servers\");\n"+
                        "               m_reader.readLine(); // whatever.\n"+
                        "               m_logger.log(java.util.logging.Level.INFO,\"App is going down.\");\n"
                        );
                // Put die logic here so we can clean up nicely.
                // Basically set the die flag on the runnable and send it a dummy
                // message. when it gets the message, the server will see the die
                // flag and well, terminate.
                it = m_netFunctions.iterator();
                m_serverCode.append(
                "                       // User pressed a key so kill all remote methods\n"+
                "                       java.net.Socket killerSocket = null;\n"
                );
                while (it.hasNext())
                {
                        fd = (FunctionDecl) it.next();
                        m_serverCode.append(
                                        "\n                     "+fd.m_id+"Inst.m_die = true;\n"+

                                        "                       killerSocket = new
java.net.Socket(java.net.InetAddress.getByName(\"localhost\"),"+fd.m_id+"Inst.m_port);\n"+
                                        "                       killerSocket.getOutputStream().flush();\n" +
                                        "                       m_logger.log(java.util.logging.Level.INFO,\"Sending kill signal to
"+fd.m_id+" on port:\"+"+fd.m_id+"Inst.m_port);\n"
                                );
                }
                m_serverCode.append(
        "               m_logger.log(java.util.logging.Level.INFO,\"Told all threads to die. See ya.\");\n" +
                        "               }\n"
                );
                m_serverCode.append("}");       // finish java file.
```

```
	}
	public String getCode(String _tabs) { return m_serverCode.toString();}
	public static String getPackageToWriteTo()
	{
		return m_packageToWriteTo;
	}
	public void setPackageToWriteTo(String _pkg)
	{
		m_packageToWriteTo = _pkg;
	}
	public static final String getAutoGenWarning()
	{
		if (m_supressAutoGenMsg)
			return "/**\n* Autogenerated on: \n*/\n";
		else
			return "/**\n* Autogenerated on: " + new Date() +"\n* Any changes you will be lost on re-
compile.\n*/\n";
	}
	public void addFunction(FunctionDecl _decl)
	{
		//m_functions.add(_decl);
		m_serverCode.append(_decl.getCode(getTabs())+"\n");
	}
	public void addNetworkFunction(FunctionDecl _decl)
	{
		m_netFunctions.add(_decl);
		m_serverCode.append(_decl.getCode(getTabs())+"\n");
	}
	/**
	 * Writes out the Socket code for each network function
	 */
	protected void writeNetworkWorker(FunctionDecl _fd)
	{
		String out =
			"		class "+_fd.m_id+"Runnable implements java.lang.Runnable\n" +
			"		{\n"+
			"			java.net.ServerSocket m_socket = null;\n" +
			"			java.net.Socket m_incomingSocket = null;\n" +

			"			boolean m_running = false;\n" +
			"			boolean m_die = false;\n"+
			"			public int m_port = -1;\n" +
			"			protected "+_fd.m_id+"Runnable()\n" +
			"			{\n" +
			"				super();\n" +
			"			}\n" +
			"			public void run()\n" +
			"			{\n"+
			"				try\n" +
			"				{\n" +
			"					m_socket = new java.net.ServerSocket(m_port);\n" +
			"					m_port = m_socket.getLocalPort();\n" +

			"				} catch (Exception e) { }\n" +

			"				m_running = true;\n" +
			"				m_logger.log(java.util.logging.Level.INFO,\"Function
"+_fd.m_id+" is waiting on port:\"+m_port);\n" +
			"				String fromClient;\n" +
			"				java.io.BufferedReader in = null;\n" +
			"				//java.io.DataInputStream dis = null;\n" +
			"				while (true)\n" +
			"				{\n" +
			"					try\n" +
			"					{\n" +
			"						m_incomingSocket = m_socket.accept();
//blocks here...\n" +
			"						//dis = new
java.io.DataInputStream(m_incomingSocket.getInputStream());\n"+
```

```
                                        "                                                                       in = new java.io.BufferedReader(new
java.io.InputStreamReader(m_incomingSocket.getInputStream()));\n"+
                                        "                                                                       while ((fromClient = in.readLine()) !=
null)\n"+
                                        "                                                                       {\n"+
                                        "                                                                               break;\n"+
                                        "                                                                       }\n"+
                                        "                                                                       if (m_die) break;\n" +
                                        //"                                                                     // TODO parse input stream here\n"+
                                                                                                                getCodeForFunctionParameters(_fd) +

                                        "                                                                       // Function call here\n" +
                                                                                                                getCodeForFunctionCall(_fd) +
                                        "                                                                       // Return message here\n" +

                                                                                                                getCodeForFunctionReturn(_fd) +

                                        "                                                               } catch (Exception e) {\n" +
                                        "
        m_logger.log(java.util.logging.Level.WARNING,\"Trouble in "+_fd.m_id+"Runnable()\",e);\n" +
                                        "                                                               } finally { try{
m_incomingSocket.close();}catch(Exception e){}}\n" +
                                        "                                               }\n" +
                                        "
        m_logger.log(java.util.logging.Level.WARNING,\""+_fd.m_id+"Runnable() told to die\");\n" +
                                        "                                       }\n"            +
                                        "               }\n";
                        m_serverCode.append(out);
        }
        /**
         * Generates code for calling the function impl. I.e. passing params, catching
         * a return value etc.
         * @param _fd
         * @return
         */
        protected String getCodeForFunctionCall(FunctionDecl _fd)
        {
                        String out = "                                                                  ";
                        // Handle return type.
                        if (NOPELType.VOID == _fd.getReturnType())
                        {
                                        out+= _fd.m_id+"(";
                        }
                        else
                        {
                                        out+=  PrimativeDecl.getJavaTypeString(_fd.getReturnType());
                                        out+=  " returnVal";
                                        if (_fd.m_returnArray)
                                        {
                                                        out+=  "[]";
                                        }
                                        out+=  "= " + _fd.m_id+"(";
                        }
                        // Handle Parameters
                        Iterator it = _fd.getParameters().iterator();
                        int numParams = _fd.getParameters().size();
                        FunctionDecl.Parameter p = null;
                        while (it.hasNext())
                        {
                                        p = (Parameter) it.next();
                                        out += (p.m_id);
                                        if (numParams-- > 1)
                                        {
                                                        out +=", ";
                                        }
                        }
                        // sew it up
                        out+= ");\n";
                        return out;
        }
```

```
	/**
	 * dump to the socket the return parameter.
	 * @param _fd
	 * @return
	 */
	protected String getCodeForFunctionReturn(FunctionDecl _fd)
	{
		String lineSpacer ="                                                                    ";
		StringBuffer out = new StringBuffer(lineSpacer);
		// VOID
		if (NOPELType.VOID == _fd.m_returnType)
		{
			// nothing to do. Just note it and bail.
			out.append("// No return parameter...easy");
			return out.toString();
		}
		// Array.
		out.append("String xmlMessage = \"<NOPELMessage>\";\n");
		if (_fd.m_returnArray)
		{
			out.append(lineSpacer+"xmlMessage +=\"<Array>\";\n");
			out.append(lineSpacer+"if (null != returnVal)\n");
			out.append(lineSpacer+"{\n");
			out.append(lineSpacer+"		for (int mm=0; mm < returnVal.length; ++mm)\n");
			out.append(lineSpacer+"		{\n");
			out.append(lineSpacer+"
	xmlMessage+=\"<"+PrimativeDecl.getJavaTypeString(_fd.m_returnType)+">\";\n");
			if ( _fd.getReturnType().equals(NOPELType.DATE))
			{
				out.append(lineSpacer+"			xmlMessage+= new
java.text.SimpleDateFormat(\"ddMMMyyyy\").format(returnVal[mm]);\n");
			} else {
				out.append(lineSpacer+"			xmlMessage+= returnVal[mm];\n");
			}
			out.append(lineSpacer+"
	xmlMessage+=\"</"+PrimativeDecl.getJavaTypeString(_fd.m_returnType)+">\";\n");
			out.append(lineSpacer+"		}\n");
			out.append(lineSpacer+"}\n");
			out.append(lineSpacer+"xmlMessage +=\"</Array>\";\n");
		} else
		{
			// We return something...
			out.append(lineSpacer+"
	xmlMessage+=\"<"+PrimativeDecl.getJavaTypeString(_fd.m_returnType)+">\";\n");
			if ( _fd.getReturnType().equals(NOPELType.DATE))
			{
				out.append(lineSpacer+"			xmlMessage+= new
java.text.SimpleDateFormat(\"ddMMMyyyy\").format(returnVal);\n");
			} else {
				out.append(lineSpacer+"			xmlMessage+= returnVal;\n");
			}
			out.append(lineSpacer+"
	xmlMessage+=\"</"+PrimativeDecl.getJavaTypeString(_fd.m_returnType)+">\";\n");
		}
		out.append(lineSpacer+"xmlMessage += \"</NOPELMessage>\";\n");

		out.append(lineSpacer+"m_incomingSocket.getOutputStream().write(xmlMessage.getBytes());\n");
		out.append(lineSpacer+"m_incomingSocket.getOutputStream().flush();\n");

		return out.toString();
	}
	/**
	 * This takes the input stream from the socket, expects an XML string
	 * that has the input parameters and declares them.
	 * @param _fd
	 * @return
	 */
	protected String getCodeForFunctionParameters(FunctionDecl _fd)
	{
		String out = "                                                    ";
```

```
                // For now just place holders
                Iterator it = _fd.getParameters().iterator();
                FunctionDecl.Parameter p = null;
                while (it.hasNext())
                {
                        p = (Parameter) it.next();
                        out +="String s = fromClient;\n";

                        if (p.m_isArray)
                        {
                                out += "\t\t\t\t"+PrimativeDecl.getJavaTypeString(p.m_type) +" "+p.m_id+"[]= null;\n";

        out+=NOPELCodeGenHelper.parseXMLForArrayVarInitialization(p.m_type,m_serverName,"fromClient",p.m_id);
                                // We need to split this out.
                                /*
                                out += "\t\t\t\t\t\t"+(PrimativeDecl.getJavaTypeString(p.m_type) +" ");
                                out +=p.m_id+"[] = null;\n";
                                out +="System.out.println(\"I did not get a chance to implement arrays from the client so
results are bogus.\");\n";
                                */
                        } else {
                                out += "\t\t\t\t\t\t"+(PrimativeDecl.getJavaTypeString(p.m_type) +" ");
                                out += NOPELCodeGenHelper.parseXMLForVarInitialization(p.m_id,p.m_type);

                                /*
                                if (NOPELType.BOOLEAN == _fd.getReturnType() ||
                                                NOPELType.DATE == _fd.getReturnType() ||
                                                NOPELType.REAL == _fd.getReturnType() ||
                                                NOPELType.INT == _fd.getReturnType() ||
                                                NOPELType.STRING == _fd.getReturnType())
                                {
                                        out +=
NOPELCodeGenHelper.parseXMLForVarInitialization(p.m_id,p.m_type);
                                } else {
                                        out +="=null;\n";
                                        out +="System.out.println(\"I did not get a chance to implement structs from
the client so results are bogus.\");\n";
                                }
                                */
                        }
                        if (it.hasNext())
                                out += "                                                                ";
                }
                return out +"\n";
        }
}
////////////////////////////////////////
////////////////////////////////////////
//StructDecl.java
//~~////////////////////////////////////
////////////////////////////////////////
package com.school.nopel.codegen.decls;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.logging.Logger;

import com.school.nopel.NOPELCodeGenerator;
import com.school.nopel.codegen.FileGenerator;
import com.school.nopel.codegen.NOPELCodeGenHelper;
import com.school.nopel.semantics.NOPELType;
/**
 */
public class StructDecl extends TypeDecl implements FileGenerator
{
        static Logger m_logger = Logger.getLogger(StructDecl.class.getName());
        /**
         * The pacakge to which we want to write.
         */
        protected String m_package = "";
```

```
	// List of properties on this decl.
	protected ArrayList m_properties = new ArrayList();
	public StructDecl()
	{
		super();
	}
	public ArrayList getProperties() {
		return m_properties;
	}
	public void addProperty(PrimativeDecl _pd) {
		m_properties.add(_pd);
	}
	/**
	 * Not too sure of the impl here. On the one hand I may
	 * want to generate a bunch of helper classes. I think
	 * inner classes will be ok...Maybe.
	 */
	public String getCode(String _tabs)
	{
		m_package = ServerDecl.getPackageToWriteTo();
		StringBuffer code = new StringBuffer();

		code.append(_tabs+"/**\n"+_tabs+"* Inner class declaration\n"+_tabs+"*/\n");
		code.append(_tabs +"public class "+m_id+"\n"+_tabs+"{\n");
		Iterator it = m_properties.iterator();
		while (it.hasNext())
		{
			code.append(_tabs +"\tpublic "+((CodeGenerator)it.next()).getCode(_tabs)+"\n");
		}
		String newTabs =_tabs + "\t";
		code.append(newTabs +"public "+m_id+"() {}\n");
		// reconstruct from an xml string
		code.append(newTabs +"public "+m_id+"(String s) {\n");
		TypeDecl p =null;
		it = m_properties.iterator();
		while (it.hasNext())
		{
			p = (TypeDecl) it.next();
			code.append(newTabs
+"\t"+NOPELCodeGenHelper.parseXMLForVarInitialization(p.getId(),p.m_type));
		}
		code.append(newTabs +"}\n");


		// Tack on a toString method so I can use this when generating XML
		code.append(newTabs +"public String toString() {\n");
		code.append(newTabs +"String out =\"\";\n");
		// We will have primatives and arrays here.
		Exportable export = null;
		it = m_properties.iterator();
		while (it.hasNext())
		{
			export = (Exportable)it.next();
			p = (TypeDecl) export;
			code.append(newTabs +"\tout +=\"<"+p.getId()+">\"+this."+p.getId()+"+\"</"+p.getId()+">\";\n");

		}
		code.append(newTabs +"return out;\n");
		code.append(_tabs +"}\n");

		code.append(_tabs +"}\n");
		return code.toString();
	}
	/**
	 *
	 */
	public void setPackage(String _pkg)
	{
		m_package = _pkg;
	}
```

```
}
////////////////////////////////////////
////////////////////////////////////////
//TypeDecl.java
//~~////////////////////////////////////
////////////////////////////////////////
package com.school.nopel.codegen.decls;

import com.school.nopel.semantics.NOPELType;
/**
 * Type Declaration
 */
public abstract class TypeDecl implements CodeGenerator
{
		NOPELType m_type;
		String m_id;
		public TypeDecl()
		{
			super();
		}
		public String getId() {
			return m_id;
		}
		public void setId(String m_id) {
			this.m_id = m_id;
		}
		public NOPELType getType() {
			return m_type;
		}
		public void setType(NOPELType _type) {
			m_type = _type;
		}
}
////////////////////////////////////////
////////////////////////////////////////
//FileGenerator.java
//~~////////////////////////////////////
////////////////////////////////////////
package com.school.nopel.codegen;
/**
 * Place holder incase I need some consistency later on.
 *
 */
public interface FileGenerator {

}
////////////////////////////////////////
////////////////////////////////////////
//NOPELCodeGenHelper.java
//~~////////////////////////////////////
////////////////////////////////////////
package com.school.nopel.codegen;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.StringTokenizer;
import java.util.logging.Level;
import java.util.logging.Logger;

import com.school.nopel.codegen.decls.FunctionDecl;
import com.school.nopel.codegen.decls.PrimativeDecl;
import com.school.nopel.semantics.NOPELType;

/**
 * Helps out in generating the code.
 */
public class NOPELCodeGenHelper
{
	static Logger m_logger = Logger.getLogger(NOPELCodeGenHelper.class.getName());
```

```
static String m_dirToGenTo = "C:/temp/autogen";
public static final String getDirToAutoGenIn()
{
	return m_dirToGenTo;
}
public static final void setDirToAutoGenIn(String _dir)
{
	m_dirToGenTo = _dir;
}
public static void ensureDirForPkgExists(String _path)
{
	File f;
	try {
		f = new File(_path);
		f.mkdirs();
	} catch (Exception e) {
		m_logger.log(Level.WARNING,"Trouble w/ path: " + _path, e);
	}
}
public static final String getDirForPkg(String _pkg)
{
	String out = "";
	if (null != _pkg)
	{
		String pkgDir = "";
		StringTokenizer st = new StringTokenizer(_pkg,".");
		while (st.hasMoreTokens())
		{
			pkgDir += ("/" + st.nextToken());
		}
		out = getDirToAutoGenIn() + pkgDir;
		ensureDirForPkgExists(out);
	} else {
		out = getDirToAutoGenIn();
	}
	return out;
}
public static void writeCodeToFile(String _fileName,String _code)
{
	FileWriter fw = null;
	File f = null;
	try {
		f = new File(_fileName);
		fw = new FileWriter( f );
		fw.write(_code.toString());
		fw.flush();
	} catch (Exception e) {
		m_logger.log(Level.SEVERE,"failed to generate file: "+ _fileName,e);
	} finally {
		if (null != fw)
			try {
				fw.close();
			} catch (IOException e) {
				//
			}
	}
}
/**
 * in both the client and the server I need to parse out xml and initialize variables
 * @param _id
 * @param _type
 * @param _xml
 * @return
 */
public static String parseXMLForVarInitialization(String _id,NOPELType _type)
{
	StringBuffer code = new StringBuffer();
	int beginLen = _id.length()+2; // +1 for the > then one more to start
	if (_type == NOPELType.INT)
	{
```

```
	code.append(_id+"="+"Integer.parseInt(s.substring(s.indexOf(\"<"+_id+">\")+"+beginLen+",s.indexOf(\"</"+_id+">\")));\
n");
		} else {
			if (_type == NOPELType.REAL)
			{

	code.append(_id+"="+"Double.parseDouble(s.substring(s.indexOf(\"<"+_id+">\")+"+beginLen+",s.indexOf(\"</"+_id+">\
")));\n");
			} else {
				if (_type == NOPELType.DATE)
				{

	code.append(_id+"="+"_initDate(s.substring(s.indexOf(\"<"+_id+">\")+"+beginLen+",s.indexOf(\"</"+_id+">\")));\n");
				} else {
					// String or struct...
					code.append(_id+"="+"new "+PrimativeDecl.getJavaTypeString(_type)+"
(s.substring(s.indexOf(\"<"+_id+">\")+"+beginLen+",s.indexOf(\"</"+_id+">\")));\n");
				}
			}
		}
		return code.toString();
	}
	public static String parseXMLForArrayVarInitialization(NOPELType _type, String _className,String
_sToParseName,String _varName)
	{
		String out = "";
		String tabs = "\t\t\t\t";
		if (NOPELType.BOOLEAN		== _type ||
			NOPELType.DATE		== _type ||
			NOPELType.REAL		== _type ||
			NOPELType.INT		== _type ||
			NOPELType.STRING	== _type)
		{
			out +=tabs+"java.util.StringTokenizer st = new java.util.StringTokenizer(ch.value,\",\");\n";
		out +=tabs+_varName+"= new "+PrimativeDecl.getJavaTypeString(_type)+"[st.countTokens()];\n";
		out +=tabs+"int i = 0;\n";
		out+=tabs+"while (st.hasMoreTokens())\n";
		out+=tabs+"{\n";
		out+=tabs+"\t"+_varName+"[i++] = ";
			if ( _type.equals(NOPELType.REAL))
			{
				out +="Double.parseDouble(st.nextToken());\n";
			} else
			{
				if ( _type.equals(NOPELType.INT))
				{
					out+= "Integer.parseInt(st.nextToken());\n";
				} else {
					if ( _type.equals(NOPELType.BOOLEAN))
					{
						out += "Boolean.valueOf(st.nextToken()).booleanValue();\n";
					} else {
						if ( _type.equals(NOPELType.DATE))
						{
							out += "_initDate(st.nextToken());\n";
						} else {
							if ( _type.equals(NOPELType.STRING))
							{

								out += "new
"+PrimativeDecl.getJavaTypeString(_type)+"(st.nextToken());";
							} else {
								// structs
								//out += "(new
"+m_serverName+"Client()).new "+PrimativeDecl.getJavaTypeString(_fd.getReturnType())+"(st.nextToken());";

								//out += "(new "+_className+"()).new
"+PrimativeDecl.getJavaTypeString(_type)+"(st.nextToken());";
```

```
                                                                                }
                                                                        }
                                                                }
                                                        }
                                                }
                                        } else {
                                                // break it up by the struct delimiter
                                                out +=tabs+"String[]
structsWeCareAbout="+_sToParseName+".split(\"</"+PrimativeDecl.getJavaTypeString(_type)+">\");\n";
                                                //out
+=tabs+"p="+_sToParseName+".split(\"</"+PrimativeDecl.getJavaTypeString(_type)+">\");\n";
                                                out +=tabs+_varName+"= new
"+PrimativeDecl.getJavaTypeString(_type)+"[structsWeCareAbout.length-1];\n";
                                                out +=tabs+"for (int k=0; k < structsWeCareAbout.length-1; ++k)\n";
                                                out +=tabs+"{\n";
                                                out+=tabs+"\t"+_varName+"[k] = ";
                                                out += "(new "+_className+"()).new
"+PrimativeDecl.getJavaTypeString(_type)+"(structsWeCareAbout[k]);\n";
                                        }
                out+=tabs+"}\n";
                return out;
                                        /*
                                        StringBuffer code = new StringBuffer();
                                        int beginLen = _id.length()+2; // +1 for the > then one more to start
                                        if (_type == NOPELType.INT)
                                        {

                code.append(_id+"="+"Integer.parseInt(s.substring(s.indexOf(\"<"+_id+">\")+"+beginLen+",s.indexOf(\"</"+_id+">\")));\
n");
                                        } else {
                                                if (_type == NOPELType.REAL)
                                                {

                code.append(_id+"="+"Double.parseDouble(s.substring(s.indexOf(\"<"+_id+">\")+"+beginLen+",s.indexOf(\"</"+_id+">\
")));\n");
                                                } else {
                                                        if (_type == NOPELType.DATE)
                                                        {

                code.append(_id+"="+"_initDate(s.substring(s.indexOf(\"<"+_id+">\")+"+beginLen+",s.indexOf(\"</"+_id+">\")));\n");
                                                        } else {
                                                                // String or struct...
                                                                code.append(_id+"="+"new "+PrimativeDecl.getJavaTypeString(_type)+"
(s.substring(s.indexOf(\"<"+_id+">\")+"+beginLen+",s.indexOf(\"</"+_id+">\")));\n");
                                                        }
                                                }
                                        }
                                        return code.toString();
                }
                public static String parseXMLForArrayVarInitialization(NOPELType _type, String _className,String
_sToParseName,String _varName)
                {
                                String out = "";
                                String tabs = "\t\t\t\t";
                                if (NOPELType.BOOLEAN                   == _type ||
                                                NOPELType.DATE                  == _type ||
                                                NOPELType.REAL                  == _type ||
                                                NOPELType.INT                   == _type ||
                                                NOPELType.STRING                == _type)
                                {
                                                out +=tabs+"java.util.StringTokenizer st = new java.util.StringTokenizer(ch.value,\",\");\n";
                                } else {
                                                // break it up by the struct delimiter
                                                out +=tabs+"java.util.StringTokenizer st = new
java.util.StringTokenizer("+_sToParseName+",\"</"+PrimativeDecl.getJavaTypeString(_type)+">\");\n";
                                }
                out +=tabs+_varName+"= new "+PrimativeDecl.getJavaTypeString(_type)+"[st.countTokens()];\n";
                out +=tabs+"int i = 0;\n";
                out+=tabs+"while (st.hasMoreTokens())\n";
                out+=tabs+"{\n";
```

```
	out+=tabs+"\t"+_varName+"[i++] = ";
			if ( _type.equals(NOPELType.REAL))
			{
				out +="Double.parseDouble(st.nextToken());\n";
			} else
			{
				if ( _type.equals(NOPELType.INT))
				{
					out+= "Integer.parseInt(st.nextToken());\n";
				} else {
					if ( _type.equals(NOPELType.BOOLEAN))
					{
						out += "Boolean.valueOf(st.nextToken()).booleanValue();\n";
					} else {
						if ( _type.equals(NOPELType.DATE))
						{
							out += "_initDate(st.nextToken());\n";
						} else {
							if ( _type.equals(NOPELType.STRING))
							{

								out += "new
"+PrimativeDecl.getJavaTypeString(_type)+"(st.nextToken());";
							} else {
								// structs
								//out += "(new "+m_serverName+"Client()).new
"+PrimativeDecl.getJavaTypeString(_fd.getReturnType())+"(st.nextToken());";

								out += "(new "+_className+"()).new
"+PrimativeDecl.getJavaTypeString(_type)+"(st.nextToken());";

							}
						}
					}
				}
			}
	out+=tabs+"}\n";
	return out;
	*/
	 }
}
////////////////////////////////////////
////////////////////////////////////////
//DeclWrapper.java
//~~////////////////////////////////////
////////////////////////////////////////
package com.school.nopel.codegen.stmts;
import com.school.nopel.codegen.decls.TypeDecl;
/**
 * Wraps primative decls and struct decls
 */
public class DeclWrapper extends StatementWrapper
{
	protected TypeDecl m_decl = null;
	public DeclWrapper(TypeDecl _decl)
	{
		super();
		m_decl = _decl;
	}
	public Object getData()
	{
		return m_decl;
		//return m_decl.getCode("") +"\n";
	}
	/**
	 * This is used for stmts that are nested e.g. in ifs
	 */
	public String toString()
	{
		return m_decl.getCode("");
```

```
	}
}
////////////////////////////////////////
////////////////////////////////////////
//StatementWrapper.java
//~~////////////////////////////////////
////////////////////////////////////////
package com.school.nopel.codegen.stmts;
/**
 * Wraps a statement.  Most statements are passed around as
 * strings of java code except decls which are objects. I
 * dont want to pass around the decls as a strings since
 * I dont know where they will live e.g. in a function or
 * globally. Yes the same can be said for the stmts.
 *
 * No Im not happy about this b/c it is a bit inconsistent. I will;
 * try to clean up later. For now though...
 */
public class StatementWrapper
{
	protected String m_code = null;
	public StatementWrapper()
	{
		super();
	}
	public StatementWrapper(String _code)
	{
		super();
		m_code = _code;
	}
	public Object getData()
	{
		return m_code;
	}
	/**
	 * This is used for stmts that are nested e.g. in ifs
	 */
	public String toString() { return m_code;}
}
////////////////////////////////////////
////////////////////////////////////////
//NOPELCompiler.java
//~~////////////////////////////////////
////////////////////////////////////////
package com.school.nopel.compiler;

import java.io.FileReader;
import java.util.Enumeration;
import java.util.Iterator;
import java.util.logging.Formatter;
import java.util.logging.Level;
import java.util.logging.LogManager;
import java.util.logging.LogRecord;
import java.util.logging.Logger;
import java.util.logging.SimpleFormatter;

import com.school.nopel.NOPELASTFactory;
import com.school.nopel.NOPELCodeGenerator;
import com.school.nopel.NOPELLexer;
import com.school.nopel.NOPELParser;
import com.school.nopel.NOPELTreeParser;
import com.school.nopel.codegen.ClientStubGenerator;
import com.school.nopel.codegen.NOPELCodeGenHelper;
import com.school.nopel.codegen.decls.ServerDecl;

import antlr.collections.AST;

public class NOPELCompiler {
	static {
		initializeVerbosity(Level.ALL);
```

```
    }
public static void main(String[] args) throws Exception
{
    //if (args.length !=3)
    if (args.length !=2)
    {
                //System.out.println("NOPELCompiler [dir to gen to (. for current)] [pacakge to gen to] [input file name]");
                System.out.println("NOPELCompiler [dir to gen to (. for current)] [input file name]");
                return;
    }

    //args[0] = "C:/temp/autogen/";
    //args[1] = "C:/temp/first.nopel";
    // Get the directory to write to (. means here.)
    String dirToGenTo = args[0]; //"C:\\temp\\autogen";
    NOPELCodeGenHelper.setDirToAutoGenIn(dirToGenTo);

    // Get the package to gen to.

    //String pkgToGenTo = args[1];
    String pkgToGenTo = null;
    ServerDecl.getInstance().setPackageToWriteTo(pkgToGenTo);
   ClientStubGenerator.getInstance().setPackageToWriteTo(pkgToGenTo);
    // Get the file to compile
    //String fileToCompile = args[2];
   String fileToCompile = args[1];

    //NOPELLexer lexer = new NOPELLexer(System.in);
    NOPELLexer lexer = new NOPELLexer(new FileReader(fileToCompile));
   NOPELParser parser = new NOPELParser(lexer);
   parser.setASTFactory(new NOPELASTFactory());
   parser.server();
   if (NOPELErrorHandler.isParseCool())
   {
    System.out.println("Parsing successful");
   }
   else
   {
    System.out.println("Compilation failed please check these probable errors:");
    Iterator it = NOPELErrorHandler.getParseErrors().iterator();
    while (it.hasNext())
    {
                System.out.println(it.next());
    }
    return;
   }
   AST t = parser.getAST();
   NOPELTreeParser tp = new NOPELTreeParser();
   tp.server(t);

   if (NOPELErrorHandler.isSemanticsCool())
   {
    System.out.println("Yeah! You can speak NOPEL (Semantics successful");
   }
   else
   {
    System.out.println("Compilation failed please check these probable errors:");
    Iterator it = NOPELErrorHandler.getSemanticErrors().iterator();
    while (it.hasNext())
    {
                System.out.println(it.next());
    }
    return;
   }
   System.out.println("Generating code to: "+ dirToGenTo);
   NOPELCodeGenerator cg = new NOPELCodeGenerator();
   String serverCode = cg.server(t);
   ClientStubGenerator csg = ClientStubGenerator.getInstance();
   csg.setServerName(ServerDecl.getServerName());
   csg.setFunctions(ServerDecl.getInstance().m_netFunctions);
```

```
        csg.setStructs(ServerDecl.getInstance().m_structs);
        String clientCode = csg.getCode("");

        // Write out the server file.
                        String fileToGen =
NOPELCodeGenHelper.getDirForPkg(ServerDecl.getPackageToWriteTo())+"/"+ServerDecl.getServerName()+".java";
                        NOPELCodeGenHelper.writeCodeToFile(fileToGen,serverCode);
                        System.out.println("Wrote server code to:" + fileToGen);

        // Write out the client file.
                        fileToGen =
NOPELCodeGenHelper.getDirForPkg(ClientStubGenerator.getPackageToWriteTo())+"/"+ServerDecl.getServerName()+"Client
.java";
                        NOPELCodeGenHelper.writeCodeToFile(fileToGen,clientCode);
                        System.out.println("Wrote client code to:" + fileToGen);

                        System.out.println("Done");
    }
    /**
         * Verbosity in the application is controlled via JavaLogger.
         *
         * @param _level
         */
        protected static void initializeVerbosity(Level _level) {
                        Logger.getLogger(".").setLevel(_level);
                        Formatter f = null;
                        f = new SimpleFormatter() {

                                        public String format(LogRecord record) {
                                                        if (null != record.getThrown()) {
                                                                        return record.getLevel() + ": " + record.getMessage()
                                                                                        + "\n" + record.getThrown().getMessage() + "\n";
                                                        } else {
                                                                        return record.getLevel() + ": " + record.getMessage()
                                                                                        + "\n";
                                                        }
                                        }
                        };
                        Enumeration names = LogManager.getLogManager().getLoggerNames();
                        Logger l;
                        while (names.hasMoreElements()) {
                                        l = ((Logger) (LogManager.getLogManager().getLogger(names
                                                        .nextElement().toString())));
                                        l.setLevel(_level);
                                        if (l.getHandlers().length > 0)
                                                        l.getHandlers()[0].setFormatter(f);
                        }
        }
}
////////////////////////////////////////
////////////////////////////////////////
//NOPELErrorHandler.java
//~~////////////////////////////////////
////////////////////////////////////////
package com.school.nopel.compiler;

import java.util.ArrayList;

import antlr.collections.AST;

/**
 * Herlper class so we can report errors to a single location.
 *
 * I separate out parse errors and semantics errors so if I become
 * smarter, I can correct probable errors and continue the compilation
 * with an assumed fix.
 */
public class NOPELErrorHandler
{
```

```
	public static ArrayList m_parseErrors = new ArrayList();
	public static ArrayList m_semanticErrors = new ArrayList();
	public static void addParseError(String _fileName, int _line, int _col,String _errorMsg)
	{
		m_parseErrors.add("Line: "+ _line+" Col: "+ _col+" : "+_errorMsg);
	}
	public static ArrayList getParseErrors()
	{
		return m_parseErrors;
	}
	public static boolean isParseCool() { return (m_parseErrors.size() < 1) ? true : false;}
	public static void clearLexerError() { m_parseErrors.clear();}


	public static void addSemanticError(String _fileName, int _line, int _col,String _errorMsg)
	{
		if (null != m_currentAST)
			m_semanticErrors.add("Line: "+ m_currentAST.getLine()+" Col: "+ m_currentAST.getColumn()+"
: "+_errorMsg);
		else
			m_semanticErrors.add("Line: ? Col: ? : "+_errorMsg);
	}
	public static ArrayList getSemanticErrors()
	{
		return m_semanticErrors;
	}
	public static boolean isSemanticsCool() { return (m_semanticErrors.size() < 1) ? true : false;}
	public static void clearSemanticErrors() { m_semanticErrors.clear();}

	private static AST m_currentAST = null;
	/**
	 * On semantics, I will hold onto the current AST and use it in case
	 * there are error messages.
	 * @param _t
	 */
	public static void holdCurrentAST(AST _t)
	{
		m_currentAST = _t;
	}

	public static void clearAllErrors()
	{
		clearLexerError();
		clearSemanticErrors();
		m_currentAST = null;
	}

}


////////////////////////////////////////
////////////////////////////////////////
//DupeIdException.java
//~~////////////////////////////////////
////////////////////////////////////////
package com.school.nopel.semantics;

public class DupeIdException extends SemanticException {
	protected static final long serialVersionUID = 0; // get rid of warning. Not serializing so dont care.
	DupeIdException() { super();}
	DupeIdException(String _string) { super(_string);}

}
////////////////////////////////////////
////////////////////////////////////////
//DupeTypeException.java
//~~////////////////////////////////////
////////////////////////////////////////
package com.school.nopel.semantics;
```

```
public class DupeTypeException extends SemanticException {
	protected static final long serialVersionUID = 0; // get rid of warning. Not serializing so dont care.
	DupeTypeException() { super();}
	DupeTypeException(String _string) { super(_string);}

}
////////////////////////////////////////
////////////////////////////////////////
//InvalidAssignmentException.java
//~~//////////////////////////////////////
//////////////////////////////////////////
package com.school.nopel.semantics;

public class InvalidAssignmentException extends SemanticException {
	protected static final long serialVersionUID = 0; // get rid of warning. Not serializing so dont care.

	InvalidAssignmentException() { super();}
	InvalidAssignmentException(String _string) { super(_string);}

}
////////////////////////////////////////
////////////////////////////////////////
//InvalidComparisonException.java
//~~//////////////////////////////////////
//////////////////////////////////////////
package com.school.nopel.semantics;

public class InvalidComparisonException extends SemanticException {
	protected static final long serialVersionUID = 0; // get rid of warning. Not serializing so dont care.
	InvalidComparisonException() { super();}
	InvalidComparisonException(String _string) { super(_string);}

}
////////////////////////////////////////
////////////////////////////////////////
//InvalidExpressionException.java
//~~//////////////////////////////////////
//////////////////////////////////////////
package com.school.nopel.semantics;

public class InvalidExpressionException extends SemanticException {
	protected static final long serialVersionUID = 0; // get rid of warning. Not serializing so dont care.

	public InvalidExpressionException() { super();}
	public InvalidExpressionException(String _string) { super(_string);}

}
////////////////////////////////////////
////////////////////////////////////////
//InvalidTypeException.java
//~~//////////////////////////////////////
//////////////////////////////////////////
package com.school.nopel.semantics;

public class InvalidTypeException extends SemanticException {
	protected static final long serialVersionUID = 0; // get rid of warning. Not serializing so dont care.
	public InvalidTypeException() { super();}
	public InvalidTypeException(String _string) { super(_string);}

}
////////////////////////////////////////
////////////////////////////////////////
//NOPELDecl.java
//~~//////////////////////////////////////
//////////////////////////////////////////
package com.school.nopel.semantics;

/**
 * Returned from a declaration. See, I had this problem with fields on structs
 * b/c properties on structs are really primatives. Now, I need to keep track of the
```

```
 * type and the id. So I created this datastructure. I think that is the only place
 * this will be used.
 */
public class NOPELDecl
{
	String m_id;
	NOPELType m_type;
	public NOPELDecl(String _id, NOPELType _type)
	{
		super();
		m_id = _id;
		m_type = _type;
	}
	public String getId() {
		return m_id;
	}
	public NOPELType getType() {
		return m_type;
	}

}
////////////////////////////////////////
////////////////////////////////////////
//NOPELEnv.java
//~~////////////////////////////////////
////////////////////////////////////////
package com.school.nopel.semantics;
import java.util.logging.Logger;


/**
 * Semantic environment. I.e. holds symbol table ans scope.
 * There should only be one so everything is static.
 */
public class NOPELEnv
{
	static Logger m_logger = Logger.getLogger(NOPELEnv.class.getName());

	protected static NOPELSymbolTable m_currentTable = null;
	/*
	protected static ArrayList m_errors = new ArrayList();

	public static void reportError(String _error)
	{
		m_errors.add(_error);
	}
	public static void reportError(Exception _error)
	{
		m_errors.add(_error.getMessage());
	}
	public static boolean areThereErrors()
	{
		return (0 != m_errors.size());
	}
	// Use this in Unit tests.
	public static Object getFirstError()
	{
		return m_errors.get(0);
	}
	public static void flushErrors()
	{
		m_errors.clear();
	}
	*/
	/**
	 * Start a new scope.
	 */
	public static void enterScope()
	{
		m_logger.entering(NOPELEnv.class.getName(),"enterScope()");
```

```
                // Create a new symbol table but push the current on as this st's parent
                m_currentTable = new NOPELSymbolTable(m_currentTable);
        };
        /**
         * Start a new scope inside a function. We need to know
         * if our return should return a value.
         */
        protected static boolean m_insideAfunctionScope = false;
        public static void enterFunctionScope()
        {
                enterScope();
                m_insideAfunctionScope = true;
        };
        public static boolean isInsideAFunction()
        {
                return m_insideAfunctionScope;
        }

        /**
         * Start a new scope inside a loop (we need to know we are in a loop
         * so we can support break and continue.
         */
        public static void enterLoopScope()
        {
                enterScope();
                m_currentTable.setCurrentScopeIsLoop(true);
        };
        /**
         * End the current scope.
         */
        public static void leaveScope()
        {
                m_logger.entering(NOPELEnv.class.getName(),"leaveScope()");

                // This class should not know about this but the symbol table will
                // manage which types are valid by modifying the set of types. This
                // call will remove those types when we go out of scope.
                m_currentTable.removeDeclaredTypes();
                m_currentTable.clearAllSymbols();
                m_currentTable.setReturnType(NOPELType.VOID);
                m_insideAfunctionScope = false;
                // Never pop the top most symbol table.
                if (null != m_currentTable.getParentTable())
                {
                        m_currentTable = m_currentTable.getParentTable();
                }
        };
        /**
         * It is ok to have break or coninue if we are in a loop.
         */
        public static boolean isOkForBreakOrContinue()
        {
                return m_currentTable.isCurrentScopeIsLoop();
        }
        public static boolean isReturnTypeCool(NOPELType _type) throws InvalidTypeException
        {
                if (!NOPELType.isOkToAssign(m_currentTable.getReturnType(),_type))
                {
                        throw new InvalidTypeException("cannot return type: "+ _type.getType() +" b/c function must
return type: "+ m_currentTable.getReturnType().getType());
                }
                return true;
        }
        /**
         * To get a handle to symbol table.
         * @return
         */
        public static NOPELSymbolTable getSymbolTable() { return m_currentTable;}
}
//////////////////////////////////////////
```

```
//////////////////////////////////////////
//NOPELStructType.java
//~~//////////////////////////////////////
//////////////////////////////////////////
package com.school.nopel.semantics;

import java.util.HashMap;

/**
 * Structs have to carry field information with them.
 *
 * Im going to keep it simple for now but I can see how at least nesting structs
 * in structs may be easy. For now, primatives only.
 */
public class NOPELStructType extends NOPELType
{
	/**
	 * Name to type map.
	 */
	HashMap m_properties = new HashMap();
	/**
	 */
	public NOPELStructType(String _type)
	{
		super(_type);
	}
	public void addProperty(String _id, NOPELType _type)
	{
		m_properties.put(_id,_type);
	}
	public NOPELType getTypeForProperty(String _id) throws SymbolNotDefinedException
	{
		NOPELType t = (NOPELType)m_properties.get(_id);
		if (null == t)
			throw new SymbolNotDefinedException("Property: "+ _id +" is not defined on this struct.");
		return t;
	}
}
//////////////////////////////////////////
//////////////////////////////////////////
//NOPELSymbolTable.java
//~~//////////////////////////////////////
//////////////////////////////////////////
package com.school.nopel.semantics;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.logging.Logger;

public class NOPELSymbolTable {
	static Logger m_logger = Logger.getLogger(NOPELSymbolTable.class.getName());
	/**
	 * The symbol table to the scope(s) above us
	 */
	private NOPELSymbolTable m_parentTable = null;
	/**
	 * The symbols defined in this scope. This contains functions
	 * and variables. I think this ok...
	 */
	private HashMap m_idToSymol = new HashMap();

	private static ArrayList m_reserved = new ArrayList();
	static {
		// load up the symbol table with all the java reserved words
		// since we are the parents
		m_reserved.add("abstract");
		m_reserved.add("assert");
		m_reserved.add("boolean");
		m_reserved.add("break");
```

```
                m_reserved.add("byte");
                m_reserved.add("case");
                m_reserved.add("catch");
                m_reserved.add("char");
                m_reserved.add("class");
                m_reserved.add("const");
                m_reserved.add("continue");
                m_reserved.add("default");
                m_reserved.add("do");
                m_reserved.add("double");
                m_reserved.add("else");
                m_reserved.add("enum");
                m_reserved.add("extends");
                m_reserved.add("final");
                m_reserved.add("finally");
                m_reserved.add("float");
                m_reserved.add("new");
                m_reserved.add("package");
                m_reserved.add("private");
                m_reserved.add("protected");
                m_reserved.add("public");
                m_reserved.add("return");
                m_reserved.add("short");
                m_reserved.add("static");
                m_reserved.add("strictfp");
                m_reserved.add("super");
                m_reserved.add("for");
                m_reserved.add("goto");
                m_reserved.add("if");
                m_reserved.add("implements");
                m_reserved.add("import");
                m_reserved.add("instanceof");
                m_reserved.add("int");
                m_reserved.add("interface");
                m_reserved.add("long");
                m_reserved.add("native");
                m_reserved.add("switch");
                m_reserved.add("synchronized");
                m_reserved.add("this");
                m_reserved.add("throw");
                m_reserved.add("throws");
                m_reserved.add("transient ");
                m_reserved.add("try");
                m_reserved.add("void");
                m_reserved.add("volatile");
                m_reserved.add("while");
        }

        protected ArrayList m_typesDeclairedInCurrentScope = new ArrayList();

        protected NOPELSymbolTable(NOPELSymbolTable _parentTable)
        {
                m_parentTable = _parentTable;
        }
        /**
         * This flag is set when in the current scope when a
         * return statement is encountered AND no more stmts
         * should follow. (this should not be set when a return
         * is encountered in an if but rather when one is encountered
         * in both an if and an else.)
         */
        protected boolean m_shouldMoreStmtsFollow = true;
        public boolean shouldMoreStmtsFollow()
        {
                return m_shouldMoreStmtsFollow;
        }
        public void setShouldMoreStmtsFollow(boolean _flag)
        {
                m_shouldMoreStmtsFollow = _flag;
        }
```

```
// TODO I have to do the same with break and continue? Less important but still...
protected boolean m_returnEncountered = false;
public boolean wasReturnEncountered()
{
	return m_returnEncountered;
}
public void setReturnEncountered(boolean _flag)
{
	m_returnEncountered = _flag;
}
/**
 * Have an indicator so we can tell if break and continue are valid.
 * This is kinda crappy to have here since this really isnt a symbol
 * table issue but I know that this gets created for each scope so
 * I dont have to worry about loops in loops. If I did it at the ENV
 * level I would not be able to tell.
 */
protected boolean m_currentScopeIsLoop = false;

protected void setCurrentScopeIsLoop(boolean _value) { m_currentScopeIsLoop = _value;}
protected boolean isCurrentScopeIsLoop() { return m_currentScopeIsLoop;}
/**
 * Similar to Loop scope, we want to know what the valid return type is.
 */
protected  NOPELType m_returnType = NOPELType.VOID;
protected  NOPELType getReturnType(){return m_returnType;}
protected void setReturnType(NOPELType _type ){m_returnType = _type;}

/**
 * This is used in unit testing.
 */
public void clearAllSymbols()
{
	m_idToSymol.clear();
}
/**
 * Useful for entering and leaving scope.
 */
protected NOPELSymbolTable getParentTable() {return m_parentTable;};
/**
 * Add a type is adding a struct. It is visible in the current scope and scopes
 * declared inside this scope.
 * @param _type
 * @return
 * @throws DupeTypeException
 */
public NOPELType addType(String _type) throws DupeTypeException, DupeIdException
{
	// Check that this type is not already being used as an identifier
	if (m_idToSymol.containsKey(_type))
	{
		throw new DupeIdException(_type +" is already used to id a variable. Cannot assign type.");
	}
	if (m_reserved.contains(_type))
	{
		throw new DupeIdException(_type +" is a reserved word.");
	}
	// will throw if the type is already defined.
	NOPELType type = NOPELType.addNewType(_type);
	m_typesDeclairedInCurrentScope.add(type);

	return type;
}
/**
 * Remove the types declared in this scope. This occurs when we leaveScope.
 * @param _type
 * @return
 * @throws DupeTypeException
 */
protected void removeDeclaredTypes()
```

```
	{
		Iterator it = m_typesDeclairedInCurrentScope.iterator();
		while (it.hasNext())
		{
			NOPELType.removeType((NOPELType)it.next());
		}
	}
	/**
	 * Add the variable as a struct type.
	 * @param _id
	 * @param _type
	 * @throws DupeIdDecl
	 */
	public void addStructVariable(String _id, NOPELType _type) throws DupeIdException
	{
		addSymbol(_id,_type,false).setReturnsStruct(true);
	}
	/**
	 * Add the variable to the current symbol table.  Error when we try to
	 * add the same symbol twice.
	 * @param _id
	 * @param _type
	 * @throws DupeIdDecl
	 */
	public void addPrimative(String _id, NOPELType _type) throws DupeIdException
	{
		addSymbol(_id,_type,false);
	}
	/**
	 * Add the variable to the current symbol table.  Error when we try to
	 * add the same symbol twice.
	 * @param _id
	 * @param _type
	 * @throws DupeIdDecl
	 */
	public void addArray(String _id, NOPELType _type) throws DupeIdException
	{
		addSymbol(_id,_type,false).setReturnsArray(true);
	}
	/**
	 * Add the function to the current symbol table.  Error when we try to
	 * add the same symbol twice.
	 * @param _id
	 * @param _type
	 * @throws DupeIdDecl
	 */
	public void addFunction(String _id, NOPELType _type) throws DupeIdException
	{
		addSymbol(_id,_type,true);
	}
	protected FunctionSymbol getFunctionSymbol(String _functionName) throws SymbolNotDefinedException
	{
		Symbol fs = getSymbol(_functionName);
		if (!(fs instanceof FunctionSymbol))
		{
			throw new SymbolNotDefinedException(_functionName+" is not a known function.");
		} else {
			return (FunctionSymbol) fs;
		}
	}
	public void addFunctionParam(String _functionName, String _id, NOPELType _type, boolean _isArray) throws
DupeIdException, SymbolNotDefinedException, SymbolNotDefinedException,SemanticException
	{
		// Make sure the function is cool
		FunctionSymbol fs = getFunctionSymbol(_functionName);
		// Add the symbol to the symbol table
		Symbol paramSymbol = addSymbol(_id,_type,false);
		paramSymbol.setReturnsArray(_isArray);
		// Add the symbol to the function
		fs.addFunctionParamSymbol(paramSymbol);
```

```
	}
	public NOPELType getFunctionParamType(String _functionName, int _index) throws DupeIdException,
SymbolNotDefinedException, SymbolNotDefinedException,SemanticException
	{
		// Make sure the function is cool
		return getFunctionSymbol(_functionName).getFunctionParamType(_index);
	}
	public void setFunctionReturnType(String _functionName, NOPELType _type,boolean _isArray) throws
SymbolNotDefinedException
	{
		// Make sure the function is cool
		FunctionSymbol fs = getFunctionSymbol(_functionName);
		fs.m_returnType = _type;
		fs.setReturnsArray(_isArray);
		 // TODO F'd up enterScope on functions & return type.
		 // This is a bit dodgy. I know this is only called from inside the
		 // funcDecl section of the TreeParser decl. I need to do this so
		 // if I come across a return stmt in the func decl I can check if
		 // it is returning the correct type. Calling leaveScope clears this
		 // value. You ask, why not just set this on enterScope()? well, I
		 // need to enter scope before I look at the parameters to the function.
		 // However, I dont know the return type until after I get past that.
		 // This may be why lang like Java have the return type up front. Anyways,
		 // whatever.
		m_returnType = _type;
	}
	public boolean isNetworkFunction(String _functionName) throws SymbolNotDefinedException
	{
		// Make sure the function is cool
		return getFunctionSymbol(_functionName).getIsNetworkFunction();
	}
	public boolean isFunction(String _functionName) throws SymbolNotDefinedException
	{
		// Make sure the function is cool
		getFunctionSymbol(_functionName);
		return true;
	}
	public int getFunctionParamCount(String _functionName) throws SymbolNotDefinedException
	{
		// Make sure the function is cool
		return getFunctionSymbol(_functionName).getNumberOfParams();
	}
	public void setAsNetworkFunction(String _functionName) throws SymbolNotDefinedException
	{
		// Make sure the function is cool
		FunctionSymbol fs = getFunctionSymbol(_functionName);
		fs.setIsNetworkFunction(true);
	}
	/**
	 * Checks the uniqueness of the symbol then adds it to the symbol table.
	 * @param _id
	 * @param _type
	 * @param _isVar
	 * @throws DupeIdDecl
	 */
	public Symbol addSymbol(String _id, NOPELType _type,boolean _isFunc) throws DupeIdException
	{
		// TODO think about maybe putting types in here as well?? May be more clear
		Symbol out = null;
		if (m_reserved.contains(_id))
		{
			throw new DupeIdException(_id +" is a reserved word.");
		}
		if (m_idToSymol.containsKey(_id) || NOPELType.isTypeDefined(_id))
		{
			throw new DupeIdException(_id +" is already defined.");
		} else {
			out = (_isFunc)? new FunctionSymbol(_id,_type) : new Symbol(_id,_type);
			m_idToSymol.put(_id,out);
		}
```

```
		return out;
	}

	public NOPELType getReturnTypeForId(String _id) throws SymbolNotDefinedException
	{
		Symbol s = getSymbol(_id);
		if (null != s)
		{
			return s.getReturnType();
		} else {
			// TODO do I want an exception here?
			m_logger.warning("Asked for type of id: "+ _id+ " but that symbol is not defined.");
			return null;
		}
	}
	protected Symbol getSymbol(String _id) throws SymbolNotDefinedException
	{
		Symbol s = (Symbol) m_idToSymol.get(_id);
		if (null == s)
		{
			// See if our parent or our up has it.
			if (null != m_parentTable)
				s = (Symbol) m_parentTable.getSymbol(_id);
			if (null == s)
				throw new SymbolNotDefinedException(_id+" is not defined.");
		}
		return s;
	}
	public boolean isIdAnArray(String _id) throws SymbolNotDefinedException
	{
		Symbol s = getSymbol(_id);
		return s.returnsArray();
	}
	/**
	 * What we store.
	 */
	class Symbol {
		protected NOPELType m_returnType	= null;
		protected boolean m_returnsArray	= false;
		protected boolean m_returnsStruct	= false;
		protected String m_name				= null;
		/**
		 * A symbol as a given name and type.
		 * @param _id
		 * @param _type
		 * @param _isVar  true means this is a var else a func
		 */
		protected Symbol(String _name, NOPELType _type)
		{
			m_returnType = _type;
			m_name = _name;
		}
		public String getName() {return m_name;}
		public NOPELType getReturnType(){return m_returnType;}
		public boolean returnsArray(){return m_returnsArray;}
		public void setReturnsArray(boolean _returnArray){m_returnsArray = _returnArray;}
		public boolean returnsStruct(){return m_returnsStruct;}
		public void setReturnsStruct(boolean _returnStruct){m_returnsStruct = _returnStruct;}
	}
	class FunctionSymbol extends Symbol
	{
		protected boolean m_isNetworkFunction = false;
		// Order list of function param types.
		ArrayList m_functionParams = new ArrayList();
		protected FunctionSymbol(String _name, NOPELType _type)
		{
			super(_name,_type);
		}
		protected int getNumberOfParams() { return m_functionParams.size();}
		protected boolean getIsNetworkFunction() { return m_isNetworkFunction;}
```

```
		protected void setIsNetworkFunction(boolean _isNetFunc) { m_isNetworkFunction = _isNetFunc;}
		protected void addFunctionParamSymbol(Symbol _s) throws SemanticException
		{
			m_functionParams.add(_s);
		}
		protected NOPELType getFunctionParamType(int _i)throws SemanticException
		{
			if ((_i > m_functionParams.size()-1) || _i < 0)
				throw new SemanticException("Function " +m_name+" does not have "+ _i+"
parameters");

			return ((Symbol) m_functionParams.get(_i)).getReturnType();
		}
	}
}
////////////////////////////////////////
////////////////////////////////////////
//NOPELType.java
//~~//////////////////////////////////////
////////////////////////////////////////
package com.school.nopel.semantics;
import java.util.HashMap;

/**
 * An Enumeration to represent all types in NOPEL
 *
 */
public class NOPELType
{
	/**
	 * Handle to all the types
	 */
	public static HashMap m_map = new HashMap();

	// system defined types
	// type_name : "boolean"| "date" | "string" |"int" | "real";
	public static final NOPELType BOOLEAN = new NOPELType("boolean");
	public static final NOPELType DATE = new NOPELType("date");
	public static final NOPELType STRING = new NOPELType("string");
	public static final NOPELType INT = new NOPELType("int");
	public static final NOPELType REAL = new NOPELType("real");

	/**
	 * Helpful for the log stmt.
	 * @param _t
	 * @return
	 */
	public static boolean isPrimativeType(NOPELType _t)
	{
		return (_t == BOOLEAN || _t == DATE || _t == STRING || _t == INT || _t == REAL);
	}
	/**
	 * for functions
	 */
	public static final NOPELType VOID = new NOPELType("void");

	// TODO think about arrays...
	/**
	 * The string representation of this type
	 */
	private String m_type = null;
	/**
	 * User defined types like structs are put into the static map on construction.
	 * @param _type
	 */
	public NOPELType(String _type)
	{
		m_type = _type;
		m_map.put(_type,this);
	}
```

```
	/**
	 * Get the types that are currently in scope. (this is important so we
	 * can support structs.)
	 */
	public static boolean isTypeDefined(String _type)
	{
		return m_map.containsKey(_type);
	}
	/**
	 * Add a new type. This is useful for structs.
	 * @param _type
	 */
	protected static NOPELType addNewType(String _type) throws DupeTypeException
	{
		if (m_map.containsKey(_type))
			throw new DupeTypeException("Type: "+ _type+" is already declaired.");
		// construction adds it to the set.
		return new NOPELStructType(_type);
	}
	/**
	 * Removing a type occurs for structs when they are dclrd locally and go out of scope
	 * @param _types
	 */
	protected static void removeType(NOPELType _type)
	{
		m_map.remove(_type.m_type);
	}
	/**
	 * Tell if the objects are the same.
	 */
	public boolean equals(Object _obj)
	{
		if (_obj instanceof String)
		{
			return _obj.toString() == m_type;
		}
		return super.equals(_obj);
	};
	/**
	 * Returns the NOPEL type for this key.
	 */
	public static NOPELType getType(String _type)
	{
		return (NOPELType)m_map.get(_type);
	}
	/**
	 * Determine if it is ok to assign the left type to the right type
	 * @param _left
	 * @param _right
	 * @return
	 */
	public static boolean isOkToAssign(NOPELType _left, NOPELType _right)
	{
		if (null == _left || null == _right)
			return false;
		if (NOPELType.VOID == _left && null == _right)
			return true;
		if (_left.equals(_right))
			return true;
		// Assign int to real
		if (_left.equals(NOPELType.REAL) &&_right.equals(NOPELType.INT))
			return true;
		// Assign anything but void to string.
		if (_left.equals(NOPELType.STRING) && !_right.equals(NOPELType.VOID))
			return true;
		return false;
	}
	public static boolean isOkForRlnOp(String _op,NOPELType lValueType,NOPELType rValueType) throws
InvalidComparisonException
	{
```

```
		if (NOPELType.INT == lValueType || NOPELType.REAL == lValueType)
		{
			if (NOPELType.INT == rValueType || NOPELType.REAL == rValueType)

			{
				return true;
			} else {
				throw new InvalidComparisonException("Right of operator: "+ _op+" is not an int or real
expression.");
			}
		} else {
				throw new InvalidComparisonException("Left of operator: "+ _op+" is not an int or real
expression.");
		}
	}
	public static boolean isOkForAndOrOp(String _op,NOPELType lValueType,NOPELType rValueType) throws
InvalidComparisonException
	{
		if (NOPELType.BOOLEAN == lValueType)
		{
			if (NOPELType.BOOLEAN == rValueType)

			{
				return true;
			} else {
				throw new InvalidComparisonException("Right of operator: "+ _op+" is not a boolean
expression.");
			}
		} else {
				throw new InvalidComparisonException("Left of operator: "+ _op+" is not a boolean
expression.");
		}
	}
	public static boolean isOkForArrayIndex(String _id, NOPELType _exprType) throws InvalidExpressionException,
SymbolNotDefinedException
	{
		boolean result = false;
		// First make sure that this identifier is an array
		if (!NOPELEnv.getSymbolTable().isIdAnArray(_id))
		{
			throw new InvalidExpressionException(_id+" is not an array but your trying to use it like one.");
		}
		// We are looking at an array ref here. make sure the expression resolves to an int.
		if (NOPELType.INT != _exprType)
			throw new InvalidExpressionException("Expr to give size of array: "+ _id+" must be an INT. It was:
"+ _exprType.getType());
		return result;
	}
	public static NOPELType getTypeForStructRef(NOPELType _t, String _structId,String _propId) throws
InvalidTypeException, SymbolNotDefinedException
	{
		// Make sure this is indeed a struct.
		 if (!(_t instanceof NOPELStructType))
			throw new InvalidTypeException(_structId+" is not a struct but youre tying to use it as one to get
property: "+ _propId);

		// ok this is a struct now make sure this is indeed a property in the struct.
		return ((NOPELStructType)_t).getTypeForProperty(_propId);
	}
	public static NOPELStructType getStructType(String _type) throws InvalidTypeException
	{
		NOPELType t = getType(_type);
		if (null == t)
			throw new InvalidTypeException(_type +"is not a struct");

		// Make sure this is indeed a struct.
		 if (!(t instanceof NOPELStructType))
			throw new InvalidTypeException(_type+" is not a struct.");

		// ok this is a struct now make sure this is indeed a property in the struct.
```

```
                return (NOPELStructType)t;
        }
        public static boolean isOkForLogStmt(NOPELType _eType) throws InvalidExpressionException
        {
                // Make sure expr is a primative
                // TODO logging primatives deviates from spec. Check
                if (!isPrimativeType(_eType))
                {
                        throw new InvalidExpressionException("Log stmt parameter must be a primative. It was: "+
_eType.getType());
                }
                return true;
        }
        /**
         * NOPEL allows a few complex add and subtract rules so we put the logic here.
         * @param _left
         * @param _right
         * @return
         */
        public static NOPELType getTypeWhenAddOrSub(NOPELType _left, NOPELType _right) throws
InvalidAssignmentException
        {
                // String +/- anything is a String.
                // TODO spec does not mention void...
                if (_left.equals(NOPELType.STRING) && !_right.equals(NOPELType.VOID))
                        return NOPELType.STRING;
                if (!_left.equals(NOPELType.VOID) && _right.equals(NOPELType.STRING))
                        return NOPELType.STRING;

                // Date and Int
                if (_left.equals(NOPELType.DATE) && _right.equals(NOPELType.INT))
                        return NOPELType.DATE;
                if (_left.equals(NOPELType.INT) && _right.equals(NOPELType.DATE))
                        return NOPELType.DATE;

                // Real and Int
                if (_left.equals(_right) && (_left.equals(NOPELType.INT)|_left.equals(NOPELType.REAL)))
                        return _left;
                if (_left.equals(NOPELType.REAL) && _right.equals(NOPELType.INT))
                        return NOPELType.REAL;
                if (_left.equals(NOPELType.INT) && _right.equals(NOPELType.REAL))
                        return NOPELType.REAL;

                throw new InvalidAssignmentException("Cant add/sub: "+ _left + "to "+ _right);
        }
        /**
         * I guess these are kinda simple but to keep the grammer file clean...
         * @param _left
         * @param _right
         * @return
         */
        public static NOPELType getTypeWhenMultiOrDiv(NOPELType _left, NOPELType _right) throws
InvalidAssignmentException
        {
                // Real and Int
                if (_left.equals(_right) && (_left.equals(NOPELType.INT)|_left.equals(NOPELType.REAL)))
                        return _left;
                if (_left.equals(NOPELType.REAL) && _right.equals(NOPELType.INT))
                        return NOPELType.REAL;
                if (_left.equals(NOPELType.INT) && _right.equals(NOPELType.REAL))
                        return NOPELType.REAL;

                throw new InvalidAssignmentException("Cant add/sub: "+ _left + "to "+ _right);
        }
        public static NOPELType getTypeForNumber(String _text)
        {
                if (-1 == _text.indexOf('.'))
                {
                        return INT;
                } else {
```

```
                                return REAL;
                        }
        }
        public String getType() {
                        return m_type;
        }
}
////////////////////////////////////////
////////////////////////////////////////
//SemanticException.java
//~~//////////////////////////////////////
////////////////////////////////////////
package com.school.nopel.semantics;

import antlr.RecognitionException;

public class SemanticException extends RecognitionException {

        protected static final long serialVersionUID = 0; // get rid of warning. Not serializing so dont care.

        SemanticException() { super();}
        SemanticException(String _string) { super(_string);}
}
////////////////////////////////////////
////////////////////////////////////////
//SymbolNotDefinedException.java
//~~//////////////////////////////////////
////////////////////////////////////////
package com.school.nopel.semantics;

public class SymbolNotDefinedException extends SemanticException {
        protected static final long serialVersionUID = 0; // get rid of warning. Not serializing so dont care.
        SymbolNotDefinedException() { super();}
        SymbolNotDefinedException(String _string) { super(_string);}

}
////////////////////////////////////////
////////////////////////////////////////
//BaseTest.java
//~~//////////////////////////////////////
////////////////////////////////////////
package com.school.nopel.tests;
import java.io.FileOutputStream;
import java.io.StringReader;
import java.util.ArrayList;
import java.util.Date;
import java.util.GregorianCalendar;
import java.util.Iterator;
import java.util.Random;

import com.school.nopel.NOPELASTFactory;
import com.school.nopel.NOPELLexer;
import com.school.nopel.NOPELParser;
import com.school.nopel.compiler.NOPELErrorHandler;
import com.school.nopel.semantics.NOPELEnv;

import junit.framework.TestCase;
/**
 * All test cases should subclass this and utilize the
 * logTestedProgram method to log statistics.
 */
public abstract class BaseTest extends TestCase
        {
        // Possible initial characters for an identifier
        final static String alphabet = "abcdefghijklmnopqrstuvwxyz_";
        final static String alphabetMore = alphabet+"0123456789";

        /**
         * The base directory for results files.
         */
```

```
	//public static String m_baseDirForTestFiles ="C:\\temp\\tests\\";
	public abstract String getBaseDirForTestFiles();
	/**
	 * True means generate test files.
	 */
	public static final boolean m_generateFiles = true;
	/**
	 * Max Number of randomly constructed servers
	 */
	public static final int m_maxRandomToRun = 10;
	/**
	 * max Number of random declarations in each randomly constructed server
	 */
	public static final int m_maxDeclarations = 10;
	/**
	 * Statistics for each test are held in these collections and
	 * logged on tear down.
	 */
	protected ArrayList m_validPassed = new ArrayList();
	protected ArrayList m_validFailed = new ArrayList();
	protected ArrayList m_invalidPassed = new ArrayList();
	protected ArrayList m_invalidFailed = new ArrayList();
	protected static java.text.SimpleDateFormat sdf = new java.text.SimpleDateFormat("ddMMMyyyy-HH.MM.ss");
		protected void basix(String _data, final boolean _expectSuccess)
		{
			if ( null != NOPELEnv.getSymbolTable())
				NOPELEnv.getSymbolTable().clearAllSymbols();
			//NOPELEnv.flushErrors();
			NOPELErrorHandler.clearAllErrors();
		//MyObserver observer = null;
			try {
		// The most basic is server { }
		StringReader m_stream = new StringReader(_data);
		NOPELLexer m_lexer = new NOPELLexer(m_stream);
		NOPELParser m_parser = new NOPELParser(m_lexer);
		m_parser.setASTFactory(new NOPELASTFactory());
		//observer = new MyObserver(_expectSuccess);
		//m_parser.addObserver(observer);
		m_parser.server();
		if (_expectSuccess)
		{
			if (!NOPELErrorHandler.isParseCool())
			{
				logTestedProgram(_data,_expectSuccess,true);
			assertTrue(NOPELErrorHandler.getParseErrors().get(0)+" on input: "+ _data,1 == 0);
			return;
			}
		} else {
			if (NOPELErrorHandler.isParseCool())
			{
				logTestedProgram(_data,_expectSuccess,false);
			assertTrue("Expected parse error did not occur on: "+_data,1 == 0);
			return;
			}
		}
		} catch (Exception e) {
				logTestedProgram(_data,_expectSuccess,!NOPELErrorHandler.isParseCool());

				assertTrue("Parsing error not expected on:"+_data+" error:"+
e.getMessage(),!_expectSuccess);
				return;
		}
		logTestedProgram(_data,_expectSuccess,!NOPELErrorHandler.isParseCool());
	}
		/*
		protected class MyObserver implements Observer
		{
			public boolean m_errorOccured = false;
			public String m_errorMessage = null;
			protected boolean m_expectSuccess = false;
```

```
                        public MyObserver(boolean _expectSuccess)
                        {
                                m_expectSuccess = _expectSuccess;
                        }
                        public void update(Observable _o, Object _msg)
                        {
                                m_errorOccured = true;
                                m_errorMessage = _msg.toString();
                        }
                };
                */
                public static class declHelper {
                        final static java.text.SimpleDateFormat sdf = new java.text.SimpleDateFormat("ddMMMyyyy");

                        protected static final int PRIM_INT             = 0;
                        protected static final int PRIM_REAL            = 1;
                        protected static final int PRIM_BOOL            = 2;
                        protected static final int PRIM_DATE            = 3;
                        public static String getRandomDecl(Random _r)
                        {
                                return getDecl(getRandomInt(1,3),_r);
                        }
                        static boolean m_intialize = true;
                        protected static String getDecl(int _type, Random _r)
                        {
                                m_intialize = !m_intialize;
                                String randomIdentifier = getRandomIdentifier();
                                switch(_type)
                                {
                                case (PRIM_INT)
                                :
                                        return "\n\tint "+ randomIdentifier +((m_intialize)? ("= "+_r.nextInt()+";") :
(";"));
                                case (PRIM_REAL)
                                :
                                        // Dont want an E. add a test.
                                        double d = (_r.nextDouble()*Math.random()*100);

                                        String s = ""+d;
                                        if (-1 != s.indexOf('E'))
                                        {
                                                d = Double.parseDouble(s.substring(0,s.indexOf('E')-1));
                                        }

                                        return "\n\treal "+ randomIdentifier +((m_intialize)? ("= "+ d +"; ") : (";"));

                                case (PRIM_BOOL)
                                :
                                        return "\n\tboolean "+ randomIdentifier +((m_intialize)? ("= "+
_r.nextBoolean()+"; ") : (";"));
                                case (PRIM_DATE)
                                :
                                        GregorianCalendar gc = new GregorianCalendar();
                                        gc.set((Math.abs(_r.nextInt()) + 2000) % 2300, _r.nextInt() %
11,Math.abs(_r.nextInt()) % 31);
                                        return "\n\tdate "+ randomIdentifier +((m_intialize)? ("=
"+sdf.format(gc.getTime())+"; ") : (";"));
                                }
                                return null;
                        }
                }
                /**
                 * Put the generated server code in the correct array. On tearDown we'll
                 * take care of this.
                 * @param _valid       // true means put this in the valid file.
                 * @param _failed      // false means this should have passed but did not.
                 */
                protected void logTestedProgram(String _program,boolean _valid,boolean _failed)
                {
                        if (_valid)
```

```
                {
                        if (_failed)
                                m_validFailed.add(_program);
                        else
                                m_validPassed.add(_program);
                } else {
                        if (_failed)
                                m_invalidPassed.add(_program);
                        else
                                m_invalidFailed.add(_program);
                }

        }
        /**
         * Write out the stats for this particular test.
         */
        protected void tearDown() throws Exception {
                super.tearDown();
                NOPELErrorHandler.clearLexerError();
                try {
                        if (m_generateFiles)
                        {
                                // Write out these results
                                int totalPassed = m_validPassed.size() + m_invalidPassed.size();
                                int totalFailed = m_validFailed.size() + m_invalidFailed.size();

                                StringBuffer sb = new StringBuffer("Test: "+ getName()+" "+totalPassed
+"/"+(totalPassed+totalFailed)+" passed\n");
                                String s = null;
                                Iterator it = m_validFailed.iterator();
                                if (totalFailed > 0)
                                {
                                        sb.append("******* These should be valid but failed.
*******\n");
                                        while (it.hasNext())
                                        {
                                                s = it.next().toString();
                                                if('\n' != s.charAt(s.length()-1))
                                                {
                                                        s = s+"\n";
                                                }
                                                sb.append(s);
                                                sb.append("**********\n");

                                        }
                                        sb.append("******* These should be invalid but passed.
*******\n");
                                        it = m_invalidFailed.iterator();
                                        while (it.hasNext())
                                        {
                                                s = it.next().toString();
                                                if('\n' != s.charAt(s.length()-1))
                                                {
                                                        s = s+"\n";
                                                }
                                                sb.append(s);

                                                sb.append("**********\n");
                                        }
                                        sb.append("******* SUCCESS BELOW*******\n");

                                }
                                it = m_validPassed.iterator();
                                sb.append("******* Valids that passed *******\n");
                                while (it.hasNext())
                                {
                                        s = it.next().toString();
                                        if('\n' != s.charAt(s.length()-1))
                                        {
                                                s = s+"\n";
```

```
                                                }
                                                sb.append(s);

                                                sb.append("*****************\n");
                                        }
                                        it = m_invalidPassed.iterator();
                                        sb.append("******* InValids that passed *******\n");
                                        while (it.hasNext())
                                        {
                                                s = it.next().toString();
                                                if(s.length() != 0 && ('\n' != s.charAt(s.length()-1)))
                                                {
                                                        s = s+"\n";
                                                }
                                                sb.append(s);
                                                sb.append("*****************\n");
                                        }

                                        FileOutputStream fos = null;
                                        try {
                                                String fileName =
getClass().getName().substring(getClass().getName().lastIndexOf('.')+1);
                                                fileName += "." + getName()+ "-"+sdf.format(new Date());
                                                if (totalFailed > 0)
                                                {
                                                        fileName += "FAILURE";
                                                }
                                                fos = new
FileOutputStream(getBaseDirForTestFiles()+fileName+((totalFailed > 0)? ".failure" :".success"));
                                                fos.write(sb.toString().getBytes());
                                                fos.flush();
                                        } catch (Exception e) {
                                                e.printStackTrace();
                                        } finally {
                                                try {
                                                        fos.close();
                                                } catch ( Throwable t) {}
                                        }

                                        m_validPassed.clear();
                                        m_validFailed.clear();
                                        m_invalidPassed.clear();
                                        m_invalidFailed.clear();
                                }
                        } catch (RuntimeException e) {
                                e.printStackTrace();
                        }
                }
                /**
                 * Just to keep the subclasses simple
                 *
                 */
                public void testBatch(String _data[],boolean _expectedResult)
                {
                        for (int i=0; i < _data.length; ++i)
                        {
                                //NOPELEnv.flushErrors();
                                NOPELErrorHandler.clearAllErrors();
                                if (null != NOPELEnv.getSymbolTable())
                                        NOPELEnv.getSymbolTable().clearAllSymbols();
                                basix(_data[i],_expectedResult);
                        }
                }
                protected static int getRandomInt(int _min, int _max)
                {
                        java.util.Random r = new java.util.Random();
                        int i=0;
                        if (_min >= 0)
                        {
                                while ( i < _min || i > _max)
```

```
                                        i = (int)Math.abs(Math.round(r.nextInt(_max+1)*Math.random()));
                        } else {
                                while ( i < _min || i > _max)
                                        i = (int)Math.round(r.nextInt() *Math.random());

                        }
                        return i ;
                }
                protected static String getRandomIdentifier()
                {
                        // Generate a random string of a random length < 20 to be nice
                        //int length = (int)Math.round((Math.abs(r.nextInt()*Math.random())) % 20) + 3;
                        int length = getRandomInt(3,20);
                        StringBuffer string = new StringBuffer();
                        // Add the first one
                        //int index = (int)Math.abs(Math.round(r.nextInt()*Math.random())) % alphabet.length();
                        int index = getRandomInt(1,alphabet.length()-1);
                        string.append(alphabet.charAt(index));
                        for (int i=0; i < length; ++i)
                        {
                                //index = Math.abs(r.nextInt()) % alphabetMore.length();
                                index = getRandomInt(1,alphabetMore.length()-1);

                                string.append(alphabetMore.charAt(index));
                        }
                        return string.toString();
                }

}
//////////////////////////////////////
//////////////////////////////////////
//ArrayTests.java
//~~//////////////////////////////////
//////////////////////////////////////
package com.school.nopel.tests.scanner;


/**
 * Tests out arrays.
 */
public class ArrayTests extends ScannerBaseTest
{
        /**
         * Test out arrays of different types valid and invalid
         */
        public void testArrays()
        {
                // Positive
                String validBasix[] = {         "server dans{boolean b[45];}",
                                                                                "server dans{int i[45];}",
                                                                                "server dans{real r[45];}",
                                                                                "server dans{date d[45];}",
                                                                                "server dans{string s[45];}"};
                testBatch(validBasix,true);

                // Negative
                String invalidBasix[] = {       "server dans{boolean b[-45];}",
                                                                                        "server dans{int i45];}",
                                                                                        "server dans{real r[45;}",
                                                                                        "server dans{date[65] foo;}",
                                                                                        "server dans{string s[45]}"};
                testBatch(invalidBasix,false);
        }

        public void testLength()
        {
                String validBasix[] = {
                                        "server dans{int size = b.length;}",
                                        "server dans{function f() returns void {while(b.length < 3) {} }}",
```

```
                                        "server dans{int i = b.length + 3;}",
                                        };
                testBatch(validBasix,true);

                String invalidBasix[] = {
                                        "server dans{b.length;}",   // this is a meaningless stmt but should it be error?
                                        };
                testBatch(invalidBasix,false);
        }
        public void testAccess()
        {
                String validBasix[] = {
                                        "server dans{boolean b[45]; boolean b1=b[6];}",
                };
                testBatch(validBasix,true);
        }
        public void testArrayOfStructs()
        {
                String validBasix[] = {
                                        "server dans{foo fooArray[45];}",
                };
                testBatch(validBasix,true);
        }
}
////////////////////////////////////////
////////////////////////////////////////
//DeclarationTests.java
//~~//////////////////////////////////////
////////////////////////////////////////
package com.school.nopel.tests.scanner;
import java.util.Random;

import com.school.nopel.tests.BaseTest;
/**
 * These are very basic tests for NOPEL. This is mainly program and
 * declaration tests.
 */
public class DeclarationTests extends ScannerBaseTest
        {
        /**
         * Declarations that are initialized with expressions
         *
         */
        public void testInitWithExpressions()
        {
                String validBasix[] = {
                                        "server dans{int d = 10;}",                                             // INT with constants

                                        "server dans{int d = 1 + 2;}",                                  // INT with addition expr
                                        "server dans{int d=1; int e = d +1;}",          // INT with addition expr & var
                                        "server dans{int d=1; int e = 1 +d;}",          // INT with addition expr & var

                                        "server dans{int d=1; int e = 2; int f = d+e;}",// INT with addition expr & var


                                        "server dans{int d= 2 * 2;}",                                   // INT with multi expr
                                        "server dans{int d= 2; int g= d* 2;}",          // INT with multi expr & var

                                        "server dans{int d= 2; int f= 2*d;}",           // INT with multi expr & var
                                        "server dans{int d=1; int e = 2; int f = d*e;}",// INT with addition expr & var

                                        // other tests with multiple levels of expressions and parens
                                        "server dans{int d=1; int e = 2; int f = d*e+2;}",
                                        "server dans{int d=1; int e = 2; int f = d*e+2*3;}",
                                        "server dans{int d=1+2+3; int e = 2+d; int f = d*e+2*3;}",
                                        "server dans{int d=(1+2+3); int e = (2+d); int f = d*(e+2)*3;}",

                                        // Test with initialization by function
                                        "server dans{int d=foo(34);}",
                };
```

```
                testBatch(validBasix,true);
        }
        /**
         * Test out date formats
         */
        public void testDates()
        {
                // TODO use calendar and test out all possible dates.
                // Positive
                String validBasix[] = {         "server dans{date d = 22Mar2004;}",
                                                                        "server dans{date d = 22MAR2004;}",
                                                                        "server dans{date d = 01JuL2004;}"};
                testBatch(validBasix,true);

                // Negative
                String invalidBasix[] = {       "server dans{date a = 33March03;}",
                                                                                "server dans{date a =
33Mar03;}",
                                                                                "server dans{date a =
33Mar2003;}",
                                                                                "server dans{date a =
13Max2003;}"};
                testBatch(invalidBasix,false);
        }
        /**
         * Test that E in the exponent is not allowed
         */
        public void testE()
        {

                basix("server "+ getRandomIdentifier()+ "{real r=90234234.2344E;}",false);
        }
        /**
         * Some basic program format that should work.
         */
        public void testBasicStructurePositive()
        {
                // Positive Tests
                String validBasix[] = {         "server dans{}",
                                        "server dans{int i = 0;}",
                                        "server dans{boolean a = true;}",
                                        "server dans{boolean a = true; int b = 2;}",
                                        "server dans{boolean a = true; int b = 2;int b2 = 2;}"};
                testBatch(validBasix,true);
        }
        /**
         * Basic program formats that should fail.
         */
        public void testBasicStructureNegative()
        {
                // Negative Tests
                String invalidBasix[] = {
                                        "",
                                        "server dans",
                                        "server {}",
                                        "server dans ()",
                                        "server dans []",
                                        "server dans { in i = 2;}",
                                        "server dans { int i = ;}",
                                        "server dans { int i = 2}",
                                        "server dans{date a = ;}"};
                testBatch(invalidBasix,false);
        }
                /**
                 * Tests out random declarations
                 */
                public void testRandomDeclarations()
                {
                        Random r = new Random();
                        int next = 0;
```

```
                                StringBuffer s = null;
                                for (int j=0; j < m_maxRandomToRun; ++j)
                                {
                                        next = Math.abs(r.nextInt()) % m_maxDeclarations;
                                        s = new StringBuffer("server "+getRandomIdentifier()+" {");

                                        for (int i=0; i < next; ++i)
                                        {
                                                s.append(BaseTest.declHelper.getRandomDecl(r));
                                        }
                                        s.append("\n}");
                                        basix(s.toString(),true);
                                }
                        }
        }
////////////////////////////////////////
////////////////////////////////////////
//ExpressionTests.java
//~~//////////////////////////////////////
////////////////////////////////////////
package com.school.nopel.tests.scanner;


/**
 * Tests out expressions.
 */
public class ExpressionTests extends ScannerBaseTest
{
        public void testVarRefExpr()
        {
                // varRef: ID( (LSQR expr RSQR) | (DOT ID))?;
                String validBasix[] = {
                                        // ID
                                        "server dans{int j =p; function f() returns void {m = 12+b;}}",
                                        "server dans{int k=m;}",
                                        // Array []
                                        "server dans{boolean b=kool[4]; }",
                                        // Struct .
                                        "server dans{int h = pool.water;}",
                };
                testBatch(validBasix,true);
        }
        public void testConstRefExpr()
        {
                // constRef : INT | REAL| STRING | DATE | booleanRef;

                String validBasix[] = {
                                        // INT
                                        "server dans{int j =12;}",
                                        "server dans{int joolp =12234234;}",
                                        // REAL
                                        "server dans{real really =12;}",
                                        "server dans{function f() returns void {realValue =12234234.23432;}}",
                                        // STRING
                                        "server dans{string k=\"Hello World\";}",
                                        "server dans{function fool() returns boolean {stringValue=\"Hi there I am string\"; }}",

                                        // DATE
                                        "server dans{date d  = 09Dec2003;}",
                                        "server dans{function f() returns int {someDate  = 09Dec2003;}}",
                                        // Boolean
                                        "server dans{boolean b = true; boolean c = false;}",
                                        "server dans{function f() returns boolean {b = true; f = false;}}",
                };
                testBatch(validBasix,true);
        }
        public void testBasicExpr()
        {
                // basicExpr : varRef | constRef | LPAREN expr RPAREN
                String validBasix[] = {
```

```
                                        // ()
                                        "server dans{function fool() returns boolean { l = (k+m); }}",
                                        "server dans{function fool() returns boolean { l = ((k+m)); }}",

                                        "server dans{function fool() returns boolean { l = ((k+(m[66]*3))); }}",
                };
                testBatch(validBasix,true);
        }
        public void testMultiExpr()
        {
                // multiExpr : basicExpr ((MULT^|DIV^) basicExpr)*
                String validBasix[] = {
                                        "server dans{int m = 1 * 3;}",
                                        "server dans{int b = 3 * m + 3*3;}",
                                        "server dans{function fool() returns boolean {bork = (3*(3+3)*3);}}",

                };
                testBatch(validBasix,true);
        }
        public void testAddExpr()
        {
                // addExpr : multiExpr ((PLUS^ | MINUS^) multiExpr)*
                String validBasix[] = {
                                        "server dans{int m = 1 + 3;}",
                                        "server dans{function fool() returns boolean {b = 3 * m + 3*3;}}",
                                        "server dans{function fool() returns boolean {bork = (3*(3+3)*3);}}",
                };
                testBatch(validBasix,true);
        }
        public void testRelExpr()
        {
                // relationalExpr : addExpr ((EQ^|NOT_EQUALS^|GT^|GTE^|LT^|LTE^) addExpr)*
                String validBasix[] = {
                                        // EQ
                                        "server dans{boolean bool = a == b;}",
                                        "server dans{boolean bewe23 = (a == b);}",
                                        "server dans{boolean bsdfv23 = a == 123;}",
                                        "server dans{boolean _bsdf = (a == 123);}",
                                        "server dans{boolean _bsdfv = 123 == b3b;}",
                                        "server dans{function fool() returns boolean {a = (133 == b56b);}}",
                                        // <
                                        "server dans{boolean bool = a < b;}",
                                        "server dans{boolean bewe23 = (a < b);}",
                                        "server dans{boolean bsdfv23 = a < 123;}",
                                        "server dans{boolean _bsdf = (a < 123);}",
                                        "server dans{boolean _bsdfv = 123 < b3b;}",
                                        "server dans{function fool() returns boolean {a = (133 < b56b);}}",

                };
                testBatch(validBasix,true);
        }
        public void testNotExpr()
        {
                // negExpr : (NOT^)* relationalExpr
                String validBasix[] = {
                                        "server dans{struct dans{boolean b = !(true);};}",
                                        "server dans{function fool() returns boolean {_dslfk = !(true);;}}",

                                        "server dans{function fool() returns boolean {__3wqrew = !(a < 34);;}}",
                                        "server dans{function fool() returns boolean {sdfsdf = !(a >= 34);}}",
                                        "server dans{boolean sdf = !(54 >= 34);}",
                };
                testBatch(validBasix,true);
        }
        public void testExprGeneral()
        {
                // expr : negExpr ((AND^|OR^) negExpr)*Positive
                String validBasix[] = {
                                        // AND
                                        "server dans{boolean bool = a & b;}",
```

```
                                        "server dans{boolean bewe23 = (a & b);}",
                                        "server dans{boolean bsdfv23 = a & true;}",
                                        "server dans{function fool() returns boolean {a = (dsfe323 & b56b);}}",
                                        // OR
                                        "server dans{boolean bool = a | b;}",
                                        "server dans{boolean bewe23 = (a | b);}",
                                        "server dans{boolean bsdfv23 = a | true;}",
                                        "server dans{function fool() returns boolean {a = (dsfe323 | b56b);}}",

                                        // Function Call Expr
                                        "server dans{int i = fool();}",
                };
                testBatch(validBasix,true);
        }
}
////////////////////////////////////////
////////////////////////////////////////
//FunctionTests.java
//~~//////////////////////////////////////
////////////////////////////////////////
package com.school.nopel.tests.scanner;


/**
 * Tests out functions.
 */
public class FunctionTests extends ScannerBaseTest
{
        public void testFunctionDecl()
        {
                testFunctionDeclGeneral(false);
        }
        public void testNetFunctionDecl()
        {
                testFunctionDeclGeneral(true);
        }
        /**
         * Test as both expressions (assignment to a variable say) and statement (standalone)
         */
        public void testFunctionCall()
        {
                // funcCallParams : LPAREN! (expr (COMMA! expr)*)? RPAREN!
                // funcCallStmt : ID funcCallParams RPAREN SEMI
                String validBasix[] = {
                                        // Parameterless
                                        "server dans{int a = f1();}",
                                        "server dans{int b = _f3();}",
                                        "server dans{function fool() returns boolean {f[33] = fr3();}}",


                                        // One
                                        "server dans{function fool() returns boolean {a = f1(i);}}",
                                        "server dans{boolean b = f1(s);}",
                                        "server dans{real s = f1(d);}",
                                        "server dans{function fool() returns boolean {s = f1(r);}}",

                                        "server dans{date d = f1(b);}",
                                        "server dans{function fool() returns boolean {f1(f);}}",
                                        "server dans{function fool() returns boolean {f1(r);};}",
                                        "server dans{function fool() returns boolean {f1(anotherFunction(44));}}",

                                        // Multiple
                                        "server dans{function fool() returns boolean {f1(i,f,444);}}",
                                        "server dans{function fool() returns boolean {f1(f(),g(),h(),i());}}",

                                        "server dans{function fool() returns boolean {_someValriable =
f1(true,22Mar2003,3.43,33);}}",

                                        "server dans{function fool() returns boolean {n = f1(f3,f2,f4,f);}}",
```

```
                                        "server dans{function fool() returns boolean {m = f1(r[0],re[1],ree[2]);}}",

                };
                testBatch(validBasix,true);

        }
        protected void testFunctionDeclGeneral( boolean _net)
        {
                String p = ((_net) ? "network" : "") + " function ";
                String validBasix[] = {
                                        // Parameterless
                                        "server dans{"+p+"f1() returns int{}}",
                                        "server dans{"+p+"_f3() returns boolean[]{int i=0;}}",
                                        "server dans{"+p+"fr3() returns myType{int i=0;b=r;}}",

                                        "server dans{"+p+"fewr() returns date {int i=0;b=r; if (e==b) { b = 3;}}}",

                                        "server dans{"+p+"fcvb() returns string[] {int i=0;b=r; while (e==b) { b = 3; if
(e==r){}}}}",
                                        "server dans{"+p+"fxcvb() returns void {while (true){};};}",

                                        // One
                                        "server dans{"+p+"f1(int i) returns int{}}",
                                        "server dans{"+p+"f1(string s) returns int{}}",
                                        "server dans{"+p+"f1(date d) returns int{}}",
                                        "server dans{"+p+"f1(real r) returns int{}}",
                                        "server dans{"+p+"f1(boolean b) returns int{}}",
                                        "server dans{"+p+"f1(userDefined f) returns int{}}",
                                        "server dans{"+p+"f1(real r[]) returns int{}}",

                                        // Multiple
                                        "server dans{"+p+"f1(int i,userDefined f,userDefined f) returns int{}}",
                                        "server dans{"+p+"f1(string s,int r, real _re,boolean d) returns int{}}",

                                        "server dans{"+p+"f1(date d1,date d2,date d3,date d4) returns int{}}",

                                        "server dans{"+p+"f1(userDefined f3,userDefined f2,userDefined f4,userDefined f)
returns int {}}",
                                        "server dans{"+p+"f1(real r[],real re[],real ree[])returns int {}}",
                                        // This is now ok
                                        "server dans{a = f1();}", // only global vars at this level.
                };
                testBatch(validBasix,true);
        }
}
////////////////////////////////////////
////////////////////////////////////////
//LexerTests.java
//~~////////////////////////////////////
////////////////////////////////////////
package com.school.nopel.tests.scanner;


/**
 * Tests specific to the Lexer
 */
public class LexerTests extends ScannerBaseTest
{

        public void testEscapedStrings()
        {
                //ESC: '\\'( 't'|             'n'|             'b' | '\\' | '"');
                //STRING : '\"' (ESC|~'\"')* '\"';

                String validBasix[] = {
                                        // Embeded
                                        "server dans{string s=\"With escape \\\\ char\";}",
                                        "server dans{string s=\"With escape \\t char\";}",
                                        "server dans{string s=\"With escape \\n char\";}",
                                        "server dans{string s=\"With escape \\b char\";}",
```

```
                                        "server dans{string s=\"With escape \\\" char\";}",

                                        // At beginning
                                        "server dans{string s=\"\\\\ char\";}",
                                        "server dans{string s=\"\\t char\";}",
                                        "server dans{string s=\"\\n char\";}",
                                        "server dans{string s=\"\\b char\";}",
                                        "server dans{string s=\"\\\" char\";}",

                                        // At end
                                        "server dans{ string s=\"With escape \\\\\";}",
                                        "server dans{ string s=\"With escape \\t\";}",
                                        "server dans{ string s=\"With escape \\n\";}",
                                        "server dans{ string s=\"With escape \\b\";}",
                                        "server dans{ string s=\"With escape \\\"\";}",

                                        // Only
                                        "server dans{ string s=\"\\\\\";}",
                                        "server dans{ string s=\"\\t\";}",
                                        "server dans{ string s=\"\\n\";}",
                                        "server dans{ string s=\"\\b\";}",
                                        "server dans{ string s=\"\\\"\";}",
                };
                testBatch(validBasix,true);
        }
}
////////////////////////////////////////
////////////////////////////////////////
//LogTests.java
//~~//////////////////////////////////////
//////////////////////////////////////
package com.school.nopel.tests.scanner;


/**
 * Tests out logging
 * logStmt :("LOG_DEBUG"^ | "LOG_INFO"^ | "LOG_WARN"^ | "LOG_ERROR"^ | "LOG_FATAL"^)
 */
public class LogTests extends ScannerBaseTest
{
        String m_loggingLevels[] = { "DEBUG","INFO","WARN","ERROR","FATAL"};
        /**
         * Test out arrays of different types valid and invalid
         */
        public void testGlobalLogging()
        {
                for (int i=0; i < m_loggingLevels.length; ++i)
                {
                        // Test with string
                        basix("server dans{LOG_"+m_loggingLevels[i]+"(\"Hi there\");}", true);

                        // Test with variable
                        basix("server dans{string s=\"My String\"; LOG_"+m_loggingLevels[i]+"(s);}", true);

                        // Test with non string type
                        basix("server dans{int i=123; LOG_"+m_loggingLevels[i]+"(i);}", true);
                        // TODO check reals that begin with .
                        // TODO check strings and arrays cast to string. Hmmm. Not sure what happens there
                        basix("server dans{real r=0.123; LOG_"+m_loggingLevels[i]+"(r);}", true);
                        basix("server dans{date d=12May2003; LOG_"+m_loggingLevels[i]+"(d);}", true);

                        // Test with expressions
                        basix("server dans{date d=12May2003; LOG_"+m_loggingLevels[i]+"(d+1);}", true);

                        basix("server dans{LOG_"+m_loggingLevels[i]+"(2+1);}", true);
                        basix("server dans{LOG_"+m_loggingLevels[i]+"(2*1);}", true);
                        basix("server dans{LOG_"+m_loggingLevels[i]+"((2*1));}", true);
                        basix("server dans{LOG_"+m_loggingLevels[i]+"((2/1)+2);}", true);
                        // TODO check somewhere for div by zero? I think java allows it. Yeah yeah run time thing
                        basix("server dans{int i=0;LOG_"+m_loggingLevels[i]+"((i/1)+i);}", true);
```

```
                        }
        }
}
////////////////////////////////////////
////////////////////////////////////////
//NOPELScannerTestSuite.java
//~~//////////////////////////////////////
//////////////////////////////////////////
package com.school.nopel.tests.scanner;

import junit.framework.Test;
import junit.framework.TestSuite;

public class NOPELScannerTestSuite extends TestSuite{
        public NOPELScannerTestSuite(String s)
        {
                        super(s);
        }
        public static Test suite() {
                        NOPELScannerTestSuite suite = new NOPELScannerTestSuite("Test for com.school.nopel.tests");
                        //$JUnit-BEGIN$
                        // Find all classes in this package that are tests
                        // and include them.
                        suite.addTestSuite(ArrayTests.class);
                        suite.addTestSuite(DeclarationTests.class);
                        suite.addTestSuite(ExpressionTests.class);
                        suite.addTestSuite(FunctionTests.class);
                        suite.addTestSuite(LexerTests.class);
                        suite.addTestSuite(LogTests.class);
                        suite.addTestSuite(StatmentTests.class);
                        suite.addTestSuite(StructTests.class);

                        //$JUnit-END$
                        return suite;
        }

}
////////////////////////////////////////
////////////////////////////////////////
//ScannerBaseTest.java
//~~//////////////////////////////////////
//////////////////////////////////////////
package com.school.nopel.tests.scanner;

import com.school.nopel.tests.BaseTest;

/**
 * All test cases should subclass this and utilize the
 * logTestedProgram method to log statistics.
 */
public class ScannerBaseTest extends BaseTest
        {
        /**
         * The base directory for results files.
         */
        public static final String m_baseDirForTestFiles ="C:\\temp\\tests\\scanner\\";
        public String getBaseDirForTestFiles() {return m_baseDirForTestFiles;};
}
////////////////////////////////////////
////////////////////////////////////////
//ScannerMain.java
//~~//////////////////////////////////////
//////////////////////////////////////////
package com.school.nopel.tests.scanner;

import java.util.Enumeration;
import java.util.logging.Formatter;
import java.util.logging.Level;
import java.util.logging.LogManager;
import java.util.logging.LogRecord;
```

```
import java.util.logging.Logger;
import java.util.logging.SimpleFormatter;

import com.school.nopel.NOPELASTFactory;
import com.school.nopel.NOPELLexer;
import com.school.nopel.NOPELParser;

public class ScannerMain {
	static {
		initializeVerbosity(Level.FINEST);
	}
   public static void main(String[] args) throws Exception
   {
      //NOPELLexer lexer = new NOPELLexer(System.in);
      NOPELLexer lexer = new NOPELLexer(System.in);
      NOPELParser parser = new NOPELParser(lexer);
      parser.setASTFactory(new NOPELASTFactory());
      parser.server();
   }
   /**
    * Verbosity in the application is controlled via JavaLogger.
    *
    * @param _level
    */
   protected static void initializeVerbosity(Level _level) {
		Logger.getLogger(".").setLevel(_level);
		Formatter f = null;
		f = new SimpleFormatter() {

			public String format(LogRecord record) {
				if (null != record.getThrown()) {
					return record.getLevel() + ": " + record.getMessage()
							+ "\n" + record.getThrown().getMessage() + "\n";
				} else {
					return record.getLevel() + ": " + record.getMessage()
							+ "\n";
				}
			}
		};
		Enumeration names = LogManager.getLogManager().getLoggerNames();
		Logger l;
		while (names.hasMoreElements()) {
			l = ((Logger) (LogManager.getLogManager().getLogger(names
					.nextElement().toString())));
			l.setLevel(_level);
			if (l.getHandlers().length > 0)
				l.getHandlers()[0].setFormatter(f);
		}
	}
}


////////////////////////////////////////
////////////////////////////////////////
//StatmentTests.java
//~~////////////////////////////////////
////////////////////////////////////////
package com.school.nopel.tests.scanner;


/**
 * Tests out arrays.
 */
public class StatmentTests extends ScannerBaseTest
{
	/**
	 * Test out while loop test condition
	 */
	public void testWhilePredicates()
	{
```

```
		// Test while perdicates
		String validBasix[] = {
				"server dans{function fool() returns boolean {while ( true ) { }}}",
				"server dans{function fool() returns boolean {while ( false ) { }}}",
				"server dans{function fool() returns boolean {while ( true ) { };}}",
				"server dans{function fool() returns boolean {while ( (a < b + 1) ) { }}}",

				"server dans{function fool() returns boolean {while ( 1 == 1 ) { }}}",
				"server dans{function fool() returns boolean {while ( a < b ) { }}}",

                                                                                };
		testBatch(validBasix,true);
	}
	/**
	 * Test out while loop contents
	 */
	public void testWhileContents()
	{
		// Test while perdicates
		String validBasix[] = {
                                        "server dans{function fool() returns boolean {while ( true ) { int i=3;}}}",
                                        "server dans{function fool() returns boolean {while ( true ) { int i=3; int j = i +i;}}}",

                                        "server dans{function fool() returns boolean {while ( true ) { int i; int j = b[34];}}}",


                                                                                };
		testBatch(validBasix,true);
	}
	/**
	 * Test out if predicates
	 */
	public void testIfPredicates()
	{
		String validBasix[] = {
                                        "server dans{function fool() returns boolean {if ( true ) {};}}",

                                        "server dans{function fool() returns boolean {if ( true ) { int i=3;}}}",
                                        "server dans{function fool() returns boolean {if ( true ) { int i=3; int j = i +i;}}}",

                                        "server dans{function fool() returns boolean {if ( true ) { int i; int j = b[34];}}}",

                                        "server dans{function fool() returns boolean {if ( true ) { int i=3;} else {}}}",
                                        "server dans{function fool() returns boolean {if ( true ) { int i=3; int j = i +i;} else { int
i=3; p=4;}}}",
                                        "server dans{function fool() returns boolean {if ( true ) { int i; int j = b[34];} else {d =
22Mar2004;}}}",

                                                                                };
		testBatch(validBasix,true);
	}
	/**
	 * Test out if contents
	 */
	public void testIfContents()
	{
		String validBasix[] = {
                                        "server dans{function fool() returns boolean {if ( true ) { int i=3;}}}",
                                        "server dans{function fool() returns boolean {if ( true ) { int i=3; int j = i +i;}}}",

                                        "server dans{function fool() returns boolean {if ( true ) { int i; int j = b[34];}}}",

                                        "server dans{function fool() returns boolean {if ( true ) { int i; int j = b[34];}}}",


                                                                                };
		testBatch(validBasix,true);
	}
	public void testArrayStructPropertyRef()
	{
```

```
                String validBasix[] = {
                        // declaration is irrelevant here in the scanner but for clarity of the test.
                        // This is chaning the 11th persons age to 30.
                        "server dans{function foo() returns void {struct person{int age=29;};person
pArray[23];pArray[10].age = 10;}}",
                        // This is chaning the 11th persons age to that of the 10th person
                        "server dans{struct person{int age=29;}; person bar[100]; function foo() returns void {bar[10].age =
bar[9].age;}}",
                };
                testBatch(validBasix,true);
        }
}
////////////////////////////////////////
////////////////////////////////////////
//StructTests.java
//~~////////////////////////////////////
////////////////////////////////////////
package com.school.nopel.tests.scanner;

import java.util.Random;

/**
 * Tests out structs.
 */
public class StructTests extends ScannerBaseTest
{
        /**
         * Test out arrays of different types valid and invalid
         */
        public void testStructs()
        {
                // Positive
                String validBasix[] = {         "server dans{struct dans{boolean b;};}",
                                                                                "server dans{struct dans{boolean b =
false;};}",
                                                                                "server dans{struct dans{int i = 7;\n};int j =
7;int k=390;\n}",
                                                                                "server dans{struct dans{real r = 3.0; int
i=90; int k;};}",
                                                                                "server dans{struct dans{boolean b =
false;};}",
                                                                                "server dans{struct dans{boolean b =
false;};}",

                };
                testBatch(validBasix,true);

                // Negative
                String invalidBasix[] = {
                                        "server dans{struct d{boolean b[34];}}" // dont allow arrays

                };
                testBatch(invalidBasix,false);
        }
        /**
         * Test some random structs.
         */
        public void testRandomStructs()
        {
                Random r = new Random();
                int next = 0;
                StringBuffer s = null;
                for (int j=0; j < m_maxRandomToRun; ++j)
                {
                        next = Math.abs(r.nextInt()) % m_maxDeclarations;
                        s = new StringBuffer("server "+getRandomIdentifier()+" { struct "+ getRandomIdentifier()+ "{");

                        for (int i=0; i < next; ++i)
                        {
                                s.append(declHelper.getRandomDecl(r));
```

```
                                }
                                s.append("\n};");
                                basix(s.toString(),true);
                        }
        }
        /**
         * Test out struct access
         *
         */
        public void testStructAccess()
        {
                        String validBasix[] = {         "server dans{int i= boring.id;}",
                        };
                        testBatch(validBasix,true);
        }
        /**
         * Test out struct instance
         *
         */
        public void testStructInstance()
        {
                        String validBasix[] = {
                        //"server dans{struct foo{boolean b = false;}; foo dInstance;}",
                        "server dans{struct cat{string name=\"NA\";}; cat mine; cat yours;}",
                        "server dans{foo dInstance;}",
                        };
                        testBatch(validBasix,true);
        }
}
//////////////////////////////////////////
//////////////////////////////////////////
//ArrayTests.java
//~~//////////////////////////////////////
//////////////////////////////////////////
package com.school.nopel.tests.semantics;


/**
 * Tests out arrays.
 */
public class ArrayTests extends SemanticsBaseTest
{
        /**
         * Test out arrays of different types valid and invalid
         */
        public void testArrays()
        {
                        // Positive
                        String validBasix[] = {         "server dans{boolean b[45];}",
                                                                                "server dans{int i[45];}",
                                                                                "server dans{real r[45];}",
                                                                                "server dans{date d[45];}",
                                                                                "server dans{string s[45];}"};
                        testBatch(validBasix,true);

                        // Negative
                        /*
                        String invalidBasix[] = {       "server dans{boolean b[-45];}",
                                                                                        "server dans{int i45];}",
                                                                                        "server dans{real r[45;}",
                                                                                        "server dans{date[65] foo;}",
                                                                                        "server dans{string s[45]}"};
                        testBatch(invalidBasix,false);
                        */

        }
        public void testArrayOfStructs()
        {
                        /* TODO I have not done semantics for struct usage yet.
                        String validBasix[] = {
```

```
                                        "server dans{struct foo{ int i=0;};\nfoo fooArray[45];}",
                };
                testBatch(validBasix,true);
                */
        }

        public void testLength()
        {
                /*
                String validBasix[] = {
                                        "server dans{int size = b.length;}",
                                        "server dans{function f() returns void {while(b.length < 3) {} }}",

                                        "server dans{int i = b.length + 3;}",
                                        };
                testBatch(validBasix,true);

                String invalidBasix[] = {
                                        "server dans{b.length;}",   // this is a meaningless stmt but should it be error?
                                        };
                testBatch(invalidBasix,false);
                */
        }
        public void testAccess()
        {
                /*
                String validBasix[] = {
                                        "server dans{boolean b[45]; boolean b1=b[6];}",
                };
                testBatch(validBasix,true);
                */
        }
}
////////////////////////////////////////
////////////////////////////////////////
//DeclarationTests.java
//~~////////////////////////////////////
////////////////////////////////////////
package com.school.nopel.tests.semantics;

/**
 * Tests the semantics of declarations. The original version of this was
 * copied from the syntax version. I then added negative tests.
 */
public class DeclarationTests extends SemanticsBaseTest
        {
        public void testAddExpressions()
        {
                testAddSubExpressions("+");
        }
        public void testSubExpressions()
        {
                testAddSubExpressions("-");
        }
        public void testMultExpressions()
        {
                testMultiAndDiv("*");
        }
        public void testDivExpressions()
        {
                testMultiAndDiv("/");
        }
        /**
         * Same scenerios shoud work with add and subtract expressions.
         * @param _operator
         */
        public void testMultiAndDiv(String _operator)
        {
                String validBasix[] = {
                                        // across same type.
```

```
                                        "server dans{int d = 10"+_operator+ "2345;}",
                                        "server dans{real r = 10.0"+_operator+ "2345.9999;}",

                                        // across different types
                                        "server dans{real d = 10"+_operator+ "2345.03;}",

                                        "server dans{real d = 1233.222"+_operator+ "10;}",

                                        // with several expressions
                                        "server dans{real d = 1233.222"+_operator+ "10"+_operator+"4324.33;}",
                                        "server dans{real d = 1233.222"+_operator+
"10"+_operator+"4324.33"+_operator+"4324.33;}",
                };
                testBatch(validBasix,true);
        }
        /**
         * Same scenerios shoud work with add and subtract expressions.
         * @param _operator
         */
        public void testAddSubExpressions(String _operator)
        {
                String validBasix[] = {
                                        // Addition across same type.
                                        "server dans{int d = 10"+_operator+ "2345;}",
                                        "server dans{real r = 10.0"+_operator+ "2345.9999;}",
                                        "server dans{string s = \"Hi there\""+_operator+ "\"Big Spender\";}",

                                        // Addition across different types
                                        "server dans{real d = 10"+_operator+ "2345.03;}",
                                        "server dans{real d = 0.12312312"+_operator+ "2345;}",


                                        "server dans{string s = \"Hi there\""+_operator+ "12345;}",

                                        "server dans{string s = \"Hi there\""+_operator+ "12345.332;}",

                                        "server dans{string s = \"Hi there\""+_operator+ "false;}",

                                        "server dans{string s = \"Hi there\""+_operator+ "true;}",

                                        "server dans{string s = \"Hi there\""+_operator+ "22Jun2004;}",

                                        "server dans{string s = 12334"+_operator+"\"Hi there\";}",

                                        "server dans{string s = false"+_operator+"\"Hi there\";}",

                                        "server dans{string s = true"+_operator+"\"Hi there\";}",

                                        "server dans{string s = 22Jun1922"+_operator+"\"Hi there\";}",


                                        "server dans{date d = 22Dec1999"+_operator+ "2;}",

                                        // with multiple expressions
                                        "server dans{date d = 22Dec1999"+_operator+ "2"+_operator+"2;}",

                };
                testBatch(validBasix,true);
        }
        public void testInitWithConstantExpressions()
        {
                String validBasix[] = {
                                        // Basics no initialization
                                        "server dans{int d;}",
                                        "server dans{real deal;}",
                                        "server dans{string sayWhat;}",
                                        "server dans{date when;}",
                                        "server dans{boolean veritas;}",
                                        // Multiple decls no initalization
                                        "server dans{int d;real deal;string sayWhat;boolean veritas;date when;}",
```

```
                                        "server dans{int d;real deal;string sayWhat2;string sayWhat;date when2;boolean
veritas;boolean veritas2;date when;}",
                                        // Basics with constant initialization
                                        "server dans{int dool=12345;}",
                                        "server dans{real deal=123423.324;}",
                                        "server dans{string sayWhat=\"Hi there pookie\";}",
                                        "server dans{date when = 22Jan2005;}",
                                        "server dans{boolean veritas=false;}",
                                        "server dans{boolean veritas=true;}",
                                        // Decls constant initalization
                                        "server dans{int d=1234;real deal=2222.334234;string sayWhat=\"Whatever I want to
say\";boolean veritas=false;date when=02Jun2005;}",
                                        "server dans{int d=44323;real deal=423423.234;string sayWhat2=\"Whatever I want to
say\";string sayWhat;date when2;boolean veritas=true;boolean veritas2=false;date when=13Jun2004;}",

                };
                testBatch(validBasix,true);
        }
                /**
                 * Tests out random declarations
                 */
                public void testRandomDeclarations()
                {
                        /*
                        Random r = new Random();
                        int next = 0;
                        StringBuffer s = null;
                        for (int j=0; j < m_maxRandomToRun; ++j)
                        {
                                next = Math.abs(r.nextInt()) % m_maxDeclarations;
                                s = new StringBuffer("server "+getRandomIdentifier()+" {");

                                for (int i=0; i < next; ++i)
                                {
                                        s.append(BaseTest.declHelper.getRandomDecl(r));
                                }
                                s.append("\n}");
                                basix(s.toString(),true);
                        }
                        */
                }
                public void testXXX(String _operator)
                {
                        String validBasix[] = {
                                        // Initialize int with size of an array.
                                        "server dans{int b[23]; int c = b.length}",
                                        // Init int with value from a struct
                                        "server dans{struct person{int age=23;};  int c = b.length}",
                                        // Init int with value from a struct in an array
                                        "server dans{int b[23]; int c = b.length}",

                        };
                        testBatch(validBasix,true);
                }
        }
////////////////////////////////////////
////////////////////////////////////////
//ExpressionTests.java
//~~//////////////////////////////////
////////////////////////////////////////
package com.school.nopel.tests.semantics;

/**
 */
public class ExpressionTests extends SemanticsBaseTest
{
        public void testRelational()
        {
                String validBasix[] = {
                                        // check relational
```

```
                                        "server dans{boolean b = 10 < 9;}",
                                        "server dans{boolean b = 10 > 9.99;}",

                                        "server dans{boolean b = 10 >= 9;}",
                                        "server dans{boolean b = 10.00 <= 9;}",
                                        "server dans{int i=2; boolean b = i > 9;}",
                                        "server dans{int i=2;int j=2; boolean b = i <= j;}",
                                        "server dans{int i=2;int j=2; boolean b = i >= j;}",
                };
                testBatch(validBasix,true);

                // Test negative
                String invalidBasix[] = {
                                        // check relational
                                        "server dans{boolean b = 10 < \"a\";}",
                                        "server dans{boolean b = 22Jun2003 > 9;}",

                };
                testBatch(invalidBasix,false);
        }
        public void testAndOr()
        {
                String validBasix[] = {
                                        // check relational
                                        "server dans{boolean b = true & false;}",
                                        "server dans{boolean b = true & (1<2);}",
                                        "server dans{boolean b = (1<2) & false;}",
                                        "server dans{boolean b = (2>33) | (1<2);}",

                };
                testBatch(validBasix,true);

                // Test negative
                String invalidBasix[] = {
                                        // check relational
                                        "server dans{boolean b = 10 & \"a\";}",
                                        "server dans{boolean b = 10 & 2;}",
                                        "server dans{boolean b = (true) & 2;}",

                };
                testBatch(invalidBasix,false);
        }
        /**
         * Same scenerios shoud work with add and subtract expressions.
         * @param _operator
         */
        public void testNegate()
        {
                String validBasix[] = {
                                        // check relational
                                        "server dans{boolean b = !true;}",
                                        "server dans{boolean b = !(1<2);}",
                                        "server dans{boolean a = false; boolean b = !(a);}",
                                        "server dans{boolean a = false; boolean b = !a;}",
                };
                testBatch(validBasix,true);

                // Test negative
                String invalidBasix[] = {
                                        // check relational
                                        "server dans{boolean b = !10;}",
                                        "server dans{boolean b = !22Jun2003;}",
                                        "server dans{int i=9;boolean b = !i;}",
                };
                testBatch(invalidBasix,false);
        }
        /**
         * Same scenerios shoud work with add and subtract expressions.
         * @param _operator
         */
```

```
	public void testFunctionExpr()
	{
		String validBasix[] = {
					// func call no params
					"server dans{function foo() returns int {int i=0; return i;}int j= foo();}",
					// func call 1 param const
					"server dans{function foo(int inP1) returns int {int i=inP1; return i;}int j= foo(23);}",
					// func call 1 param var
					"server dans{function foo(int inP1) returns int {int i=inP1; return i;}int g=23; int j=
foo(g);}",
					// func call 1 param func call
					"server dans{function bar() returns int {returns 12;}\nfunction foo(int inP1) returns int
{int i=inP1; return i;}\nint j= foo(bar());}",
		};
		testBatch(validBasix,true);

		// Test negative
		String invalidBasix[] = {
					// bad type assignment
					"server dans{function foo() returns void {}\nint j= foo();}",
		};
		testBatch(invalidBasix,false);
	}
}
//////////////////////////////////////
//////////////////////////////////////
//FunctionTests.java
//~~//////////////////////////////////
//////////////////////////////////////
package com.school.nopel.tests.semantics;


/**
 * Tests out functions.
 */
public class FunctionTests extends SemanticsBaseTest
{
	public void testFunctionsAndGlobals()
	{
		String validBasix[] = {
					// Test calling a function that uses a global variable.
					"server dans{string myName= \"Dan\"; function sayHiDan() returns void { string s =
myName;}}",
		};
		testBatch(validBasix,true);

		// negative
		/*
		String invalidBasix[] = {
					// Test calling a function in a function that dne
					"server dans{function f1() returns int{}}",

		};
		testBatch(invalidBasix,true);
		*/
	}
	public void testMultipleFunctions()
	{
		String validBasix[] = {
					// Test calling a function from in a function
					"server dans{function f1() returns int{return 1;}function f2() returns int{return f1();}}",

					// Test recursion...this is infinite but still allowable...

					"server dans{function f1() returns int{return f1();}}",

					// Test implicit cast
					"server dans{function f2g() returns int { return 1234; };\nfunction f1() returns real {return
f2g();}}",
```

```
                                        "server dans{function f2h() returns int { return 1234; };\nfunction f1() returns string
{return f2h();}}",
                                        "server dans{function f2i() returns real { return 1234.22; };\nfunction f1() returns string
{return f2i();}}",
                                        "server dans{function f2j() returns date { return 22Mar2005; };\nfunction f1() returns
string {return f2j();}}",

                                        // Test network functions call non networked
                                        "server dans{function f2() returns date { return 22Mar2005; };\nnetwork function f1()
returns string {return f2();}}",
                        };
                        testBatch(validBasix,true);

                        // negative

                        String invalidBasix[] = {
                                        // Test calling a function in a function that dne
                                        "server dans{function f1() returns int{return boo();}}",

                                        // Test calling a function that has diff return type
                                        "server dans{function f2a() returns void { };\nfunction f1() returns int{return f2a();}}",
                                        "server dans{function f2b() returns string { return \"Hi there\"; };\nfunction f1() returns
int{return f2b();}}",
                                        "server dans{function f2c() returns void { };\nfunction f1() returns string {return
f2c();}}",
                                        // in valid cast
                                        /* TODO PBLM FUNCTIONS NOT RETURNING WHEN THEY SHOULD!!!! */
                                        "server dans{function f2d() returns real { };\nfunction f1() returns int {return f2d();}}",

                                        "server dans{function f2e() returns date { };\nfunction f1() returns int {return f2e();}}",

                                        // Test calling network functions
                                        "server dans{network function f3() returns date { return 22Mar2005; };\nnetwork
function f1() returns string {return f3();}}",
                                        "server dans{network function f4() returns date { return 22Mar2005; };\nfunction f1()
returns string {return f4();}}",
                        };
                        testBatch(invalidBasix,false);


        }
        public void testFunctionDecl()
        {
                        testFunctionParamsGeneral(false);
        }
        public void testNetFunctionDecl()
        {
                        testFunctionParamsGeneral(true);
        }
        protected void testFunctionParamsGeneral( boolean _net)
        {
                        String p = ((_net) ? "network" : "") + " function ";
                        String validBasix[] = {
                                        // Parameterless
                                        "server dans{"+p+"f1() returns int{return 1;}}",
                                        "server dans{"+p+"_f3() returns boolean[]{int i=0; return false;}}",

                                        "server dans{int b=22; int r=3; struct myType{int i=0;}; "+p+"fr3() returns myType{int
i=0;b=r; myType inst; return inst;}}",
                                        "server dans{"+p+"fewr() returns date {int i=0;int r=0;int e=2;int b=r; if (e==b) { b =
3;}return 22Jul2111;}}",
                                        "server dans{"+p+"fcvb() returns string[] {int i=0;int r=0;int e=2;int b=r; while (e==b) {
b = 3; if (e==r){}}string s[3]; return s;}}",
                                        "server dans{"+p+"fxcvb() returns void {while (true){};};}",

                                        // One
                                        "server dans{"+p+"f1(int i) returns int{return 1;}}",
                                        "server dans{"+p+"f12(string s) returns int{return 1;}}",

                                        "server dans{"+p+"f13(date d) returns int{return 1;}}",
```

```
                                        "server dans{"+p+"f14(real r) returns int{return 1;}}",
                                        "server dans{"+p+"f15(boolean b) returns int{return 1;}}",

                                        "server dans{struct userDefined{int i=0;}; "+p+"f16(userDefined f) returns int{return
1;}}",
                                        "server dans{"+p+"f17(real r[]) returns int{return 1;}}",

                                        // Multiple
                                        "server dans{struct userDefined {};"+p+"f21(int i,userDefined f,userDefined f1) returns
int{return 1;}}",
                                        "server dans{"+p+"f22(string s,int r, real _re,boolean d) returns int{return 1;}}",

                                        "server dans{"+p+"f23(date d1,date d2,date d3,date d4) returns int{return 1;}}",

                                        "server dans{struct userDefined{int i=0;}; "+p+"f24(userDefined f3,userDefined
f2,userDefined f4,userDefined f) returns int {return 1;}}",
                                        "server dans{"+p+"f25(real r[],real re[],real ree[])returns int {return 1;}}",
                };
                testBatch(validBasix,true);
        }
}
////////////////////////////////////////
////////////////////////////////////////
//LogTests.java
//~~////////////////////////////////////
////////////////////////////////////////
package com.school.nopel.tests.semantics;


/**
 * Tests out logging
 * logStmt :("LOG_DEBUG"^ | "LOG_INFO"^ | "LOG_WARN"^ | "LOG_ERROR"^ | "LOG_FATAL"^)
 */
public class LogTests extends SemanticsBaseTest
{
        String m_loggingLevels[] = { "DEBUG","INFO","WARN","ERROR","FATAL"};
        /**
         * Test out logging at the global leve.
         */
        public void testGlobalLoggingPositive()
        {
                for (int i=0; i < m_loggingLevels.length; ++i)
                {
                        // Test with string
                        basix("server dans{LOG_"+m_loggingLevels[i]+"(\"Hi there\");}", true);

                        // Test with variable
                        basix("server dans{string s=\"My String\"; LOG_"+m_loggingLevels[i]+"(s);}", true);

                        // Test with non string type
                        basix("server dans{int i=123; LOG_"+m_loggingLevels[i]+"(i);}", true);
                        // TODO check reals that begin with .
                        // TODO check strings and arrays cast to string. Hmmm. Not sure what happens there
                        basix("server dans{real r=0.123; LOG_"+m_loggingLevels[i]+"(r);}", true);
                        basix("server dans{date d=12May2003; LOG_"+m_loggingLevels[i]+"(d);}", true);

                        // Test with expressions
                        basix("server dans{date d=12May2003; LOG_"+m_loggingLevels[i]+"(d+1);}", true);

                        basix("server dans{LOG_"+m_loggingLevels[i]+"(2+1);}", true);
                        basix("server dans{LOG_"+m_loggingLevels[i]+"(2*1);}", true);
                        basix("server dans{LOG_"+m_loggingLevels[i]+"((2*1));}", true);
                        basix("server dans{LOG_"+m_loggingLevels[i]+"((2/1)+2);}", true);
                        // TODO check somewhere for div by zero? I think java allows it. Yeah yeah run time thing
                        basix("server dans{int i=0;LOG_"+m_loggingLevels[i]+"((i/1)+i);}", true);

                        // Test logging an element of an array
                        basix("server dans{string arr[23]; LOG_"+m_loggingLevels[i]+"(arr[0]);}", true);

                        // Test logging an field on a struct
```

```
                                basix("server dans{struct ds{int i=0;}; ds dsInst; LOG_"+m_loggingLevels[i]+"(dsInst.i);}", true);

                                // TODO this violates spec. We are logging an array. I'd like to have this though...
                                // TODO also allow logging of structs. I would have to carry along struct property names then. But
                                // do that later as it is a nice to have...unless it makes code gen easier.
                                basix("server dans{string arr[23]; LOG_"+m_loggingLevels[i]+"(arr);}", true);

                        }
        }
        public void testGlobalLoggingNegative()
        {
                        for (int i=0; i < m_loggingLevels.length; ++i)
                        {
                                // Test trying to log a struct
                                basix("server dans{struct ds{int i=0;}; ds dsInst; LOG_"+m_loggingLevels[i]+"(dsInst);}", false);

                        }
        }
        /**
         * Test out logging in different scopes
         */
        public void testScopedLogging()
        {
                        for (int i=0; i < m_loggingLevels.length; ++i)
                        {
                                // Test in func
                                basix("server dans{function f() returns void {LOG_"+m_loggingLevels[i]+"(\"Hi there\");};}",
true);
                                // Test is network func
                                basix("server dans{network function f() returns void {LOG_"+m_loggingLevels[i]+"(\"Hi
there\");};}", true);
                                // Test in loop
                                basix("server dans{function f() returns void {while(true){LOG_"+m_loggingLevels[i]+"(\"Hi
there\");}};}", true);
                                // Test in if
                                basix("server dans{function f() returns void {if(true){LOG_"+m_loggingLevels[i]+"(\"Hi
there\");}};}", true);
                                // Test in else
                                basix("server dans{function f() returns void {if(true){}else{LOG_"+m_loggingLevels[i]+"(\"Hi
there\");}};}", true);

                                // Test is struct. Should fail
                                // TODO check out logging in struct. This is weird. It causes parse error but I dont seem
                                // to do anything with it so semantic check happens anyways.
                                //basix("server dans{struct ds{int i=0;LOG_"+m_loggingLevels[i]+"(\"Hi there\");};", false);
                        }
        }
}
////////////////////////////////////////
////////////////////////////////////////
//NOPELSemanticsTestSuite.java
//~~////////////////////////////////////
////////////////////////////////////////
package com.school.nopel.tests.semantics;

import com.school.nopel.semantics.NOPELEnv;
import com.school.nopel.tests.scanner.LogTests;

import junit.framework.Test;
import junit.framework.TestResult;
import junit.framework.TestSuite;

public class NOPELSemanticsTestSuite {

        public static Test suite() {
                NOPELSemanticsTestSuite instance = new NOPELSemanticsTestSuite();
                TestSuite suite = instance.new MyTestSuite(
                                        "Test for com.school.nopel.tests.semantics");
                //$JUnit-BEGIN$
                suite.addTestSuite(ArrayTests.class);
```

```
                suite.addTestSuite(DeclarationTests.class);
                suite.addTestSuite(ExpressionTests.class);
                suite.addTestSuite(FunctionTests.class);
                suite.addTestSuite(LogTests.class);
                suite.addTestSuite(ReturnTests.class);
                suite.addTestSuite(StatmentTests.class);
                suite.addTestSuite(StructTests.class);

                //$JUnit-END$
                return suite;
        }
        public class MyTestSuite extends TestSuite
        {
                public MyTestSuite(String _s) { super(_s);}
                public void runTest(Test test, TestResult result) {
                        if (null != NOPELEnv.getSymbolTable())
                                NOPELEnv.getSymbolTable().clearAllSymbols();
                        super.runTest(test, result);
                }

        }

}
////////////////////////////////////////
////////////////////////////////////////
//ReturnTests.java
//~~////////////////////////////////////
////////////////////////////////////////
package com.school.nopel.tests.semantics;


/**
 * Test out the return stmt. It is quite particular. More details? Well
 * the return stmt is complex on if stmts with elses. E.g. in a function
 * that has an if and an else, if both the if and else return then no
 * stmts should follow. And the like. See tests below.
 */
public class ReturnTests extends SemanticsBaseTest
{
        public void testGlobalReturn()
        {
                String validBasix[] = {
                                        // Test a return at the global level.
                                        "server dans{return;}",
                                        "server dans{int i=0; string s=\"Hi\";return;}",
                                        // Can declare functions after return
                                        "server dans{int i=0; return; function foo() returns void {};}",

                                        "server dans{int i=0; return; network function foo() returns void {};}",

                                        "server dans{int i=0; return; network function foo() returns void {};function bar() returns
void {};}",

                };
                testBatch(validBasix,true);

                // negative
                String invalidBasix[] = {
                                        // Servers dont return anything.
                                        "server dans{return true;}",

                                        "server dans{return 1;}",
                                        "server dans{return \"abc\";}",
                                        // Cant declare after the return
                                        "server dans{int i=0; string s=\"Hi\";return;int i=0;}",
                };
                testBatch(invalidBasix,false);
        }
        public void testFunctionsSimple()
        {
```

```
                String validBasix[] = {
                                // VOID
                                "server dans{function f1() returns void {};}",
                                "server dans{function f1() returns void {return;};}",
                                "server dans{function f2() returns void {int i=0; string s=\"Sammy\";return;};}",

                                "server dans{function f3() returns void {int i=0; string s=\"Sammy\";return;};}",


                                // Return at both global and functional level
                                "server dans{int i=10; return; function f4() returns void {int i=0; string
s=\"Sammy\";return;};}",
                                // Multiple functions
                                "server dans{int i=10; return; function f5() returns void {int i=0; string
s=\"Sammy\";return;};}function f6() returns void {};}",
                };
                testBatch(validBasix,true);

                // negative
                String invalidBasix[] = {
                                // decl after return
                                "server dans{function f11() returns void {int i=0; return; string s=\"Sammy\";return;};}",
                                // stmt after return
                                "server dans{function f22() returns void {int i=0; return; f22();};}",

                                "server dans{function f33() returns void {int i=0; return; i = i +1;}",

                                // Dupe returns
                                "server dans{function f44() returns void {return; return;}",

                                "server dans{function f55() returns void {int i=0; return; return;}",
                                // Missing return
                                "server dans{function f66() returns int {int i=0;}",
                                "server dans{function f77() returns string {string s=\"Hi there\";}",


                };
                testBatch(invalidBasix,false);
        }
        public void testIfs()
        {
                String validBasix[] = {
                                // At the global level if with no else returns
                                "server dans{int i=0; if (i < -23) { return; } string s=\"should see me\";}",
                                // At the global level if with no else does not return
                                "server dans{int i=0; if (i < -23) {} string s=\"should see me\";}",

                                // At the global level if with else both return
                                "server dans{int i=0; if (i < -23) {return;} else { return;};}",

                                // At the global level if with else if return
                                "server dans{int i=0; if (i < -23) {return;} else { int j=100;};string s=\"should see me\";
return;}",
                                // At the global level if with else return
                                "server dans{int i=0; if (i < -23) {} else { int j=100;return;};string s=\"should see me\";
return;}",
                                // At the global level if with neither return
                                "server dans{int i=0; if (i < -23) {} else { int j=100;};string s=\"should see me\";
return;}",
                                // Global nested ifs
                };
                testBatch(validBasix,true);

                // negative
                String invalidBasix[] = {
                                // if and else both return. No stmt afterwards
                                "server dans{int i=0; if (i < -23) {return;} else { return;} string s=\"should error\";}",

                                // if and else stmts after return.
                };
```

```
                testBatch(invalidBasix,false);
        }
}
////////////////////////////////////////
////////////////////////////////////////
//SemanticsBaseTest.java
//~~////////////////////////////////////
////////////////////////////////////////
package com.school.nopel.tests.semantics;
import java.io.StringReader;

import antlr.collections.AST;

import com.school.nopel.NOPELASTFactory;
import com.school.nopel.NOPELLexer;
import com.school.nopel.NOPELParser;
import com.school.nopel.NOPELTreeParser;
import com.school.nopel.compiler.NOPELErrorHandler;
import com.school.nopel.semantics.NOPELEnv;
import com.school.nopel.tests.BaseTest;
/**
 * All test cases should subclass this and utilize the
 * logTestedProgram method to log statistics.
 */
public class SemanticsBaseTest extends BaseTest
        {
        /**
         * The base directory for results files.
         */
        public static final String m_baseDirForTestFiles ="C:\\temp\\tests\\semantics\\";
        public String getBaseDirForTestFiles() {return m_baseDirForTestFiles;};

                protected void basix(String _data, final boolean _expectSuccess)
                {
                        try {
                        if (null != NOPELEnv.getSymbolTable())
                                NOPELEnv.getSymbolTable().clearAllSymbols();
                                //NOPELEnv.flushErrors();
                                NOPELErrorHandler.clearAllErrors();
                // The most basic is server {}
                StringReader m_stream = new StringReader(_data);
                NOPELLexer m_lexer = new NOPELLexer(m_stream);
                NOPELParser m_parser = new NOPELParser(m_lexer);
                m_parser.setASTFactory(new NOPELASTFactory());
                m_parser.server();
                AST t = m_parser.getAST();
          NOPELTreeParser tp = new NOPELTreeParser();
          tp.server(t);
                if (_expectSuccess)
                {
                        //if (NOPELEnv.areThereErrors())
                        if (!NOPELErrorHandler.isSemanticsCool())
                        {
                                logTestedProgram(_data,_expectSuccess,true);
                        //assertTrue(NOPELEnv.getFirstError()+" on input: "+ _data,1 == 0);
                                assertTrue(NOPELErrorHandler.getSemanticErrors().get(0)+" on input: "+ _data,1 == 0);

                        return;
                        }
                } else {
                        //if (!NOPELEnv.areThereErrors())
                        if (NOPELErrorHandler.isSemanticsCool())
                        {
                                logTestedProgram(_data,_expectSuccess,false);
                        assertTrue("Expected semantic error did not occur on: "+_data,1 == 0);
                        return;
                        }
                }
                } catch (Exception e) {
                                logTestedProgram(_data,_expectSuccess,true);
```

```
                                        assertTrue("Semantic error not expected on:"+_data+" error:"+
e.getMessage(),!_expectSuccess);
                                        return;
                        }
                        logTestedProgram(_data,_expectSuccess,!NOPELErrorHandler.isSemanticsCool());
        }
                        protected void tearDown() throws Exception
                        {
                                super.tearDown();
                                if (null != NOPELEnv.getSymbolTable())
                                        NOPELEnv.getSymbolTable().clearAllSymbols();
                        }
}
////////////////////////////////////////
////////////////////////////////////////
//StatmentTests.java
//~~////////////////////////////////////
////////////////////////////////////////
package com.school.nopel.tests.semantics;


/**
 * Tests out arrays.
 */
public class StatmentTests extends SemanticsBaseTest
{
        public void testAssignment()
        {
                        // Test while perdicates
                        String validBasix[] =
                        {
                                "server dans{int one=1; one = 10; one = one + one;}",
                        };
                        testBatch(validBasix,true);

        }
        /**
         * Test out while loop test condition
         */
        public void testWhilePredicates()
        {
                        // Test while perdicates
                        String validBasix[] = {
                                "server dans{while ( true ) { }}}",
                                "server dans{while ( false ) { }}",
                                "server dans{while ( true ) { };}",
                                "server dans{int a=1; int b=2; while ( (a < b + 1) ) { }}",

                                "server dans{while ( 1 == 1 ) { }}",
                                "server dans{int a=1;int b=2;while ( a < b ) { }}",
                                                                                        };
                        testBatch(validBasix,true);
        }
        /**
         * Test out while loop contents
         */
        public void testWhileContents()
        {
                        /*
                        // Test while perdicates
                        String validBasix[] = {
                                        "server dans{function fool() returns boolean {while ( true ) { int i=3;}}}",
                                        "server dans{function fool() returns boolean {while ( true ) { int i=3; int j = i +i;}}}",

                                        "server dans{function fool() returns boolean {while ( true ) { int i; int j = b[34];}}}",


                                                                                        };
                        testBatch(validBasix,true);
                        */
```

```
	}
	/**
	 * Test out if predicates
	 */
	public void testIfPredicates()
	{
		String validBasix[] = {
					"server dans{if ( true ) {};}}",
					"server dans{if ( false ) { int i=3;}}}",
					"server dans{int a=1; int b=3; if ( a == b ) { int i=3; int j = i +i;}}",

		};
		testBatch(validBasix,true);
	}
	/**
	 * Test out if contents
	 */
	public void testIfContents()
	{
		/*
		String validBasix[] = {
					"server dans{function fool() returns boolean {if ( true ) { int i=3;}}}",
					"server dans{function fool() returns boolean {if ( true ) { int i=3; int j = i +i;}}}",

					"server dans{function fool() returns boolean {if ( true ) { int i; int j = b[34];}}}",

					"server dans{function fool() returns boolean {if ( true ) { int i; int j = b[34];}}}",


		                                                                                        };
		testBatch(validBasix,true);
		*/
	}
	public void testArrayStructPropertyRef()
	{
		/*
		String validBasix[] = {
			// declaration is irrelevant here in the scanner but for clarity of the test.
			// This is chaning the 11th persons age to 30.
			"server dans{function foo() returns void {struct person{int age=29;};person
pArray[23];pArray[10].age = 10;}}",
			// This is chaning the 11th persons age to that of the 10th person
			"server dans{struct person{int age=29;}; person bar[100]; function foo() returns void {bar[10].age =
bar[9].age;}}",
		};
		testBatch(validBasix,true);
		*/
	}
}
//////////////////////////////////////////
//////////////////////////////////////////
//StructTests.java
//~~//////////////////////////////////////
//////////////////////////////////////////
package com.school.nopel.tests.semantics;

import java.util.Random;

import com.school.nopel.compiler.NOPELErrorHandler;
import com.school.nopel.semantics.NOPELEnv;


/**
 * Tests out structs.
 */
public class StructTests extends SemanticsBaseTest
{
	/**
	 * Test out arrays of different types valid and invalid
	 */
```

```
	public void testStructDecl()
	{
		String validBasix[] = {
					// No properties
					"server dans{};}",
					// Single no initialization
					"server dans{struct dans1{boolean b5tgs;};}",
					"server dans{struct dans1{int b34twre;};}",
					"server dans{struct dans1{real b34rewf;};}",
					"server dans{struct dans1{string b34fds;};}",
					"server dans{struct dans1{date bsdfsadf;};}",
					// Single with init
					"server dans{struct dans1{boolean b5tgs=false;};}",
					"server dans{struct dans1{int b34twre=12345;};}",
					"server dans{struct dans1{real b34rewf=454.054;};}",
					"server dans{struct dans1{string b34fds=\"Hi there world\";};}",
					"server dans{struct dans1{date bsdfsadf=01APR2006;};}",
					// Multiple no init
					"server dans{struct dans1{date bsdfsadf;string b34fds;real b34rewf;int b34twre;boolean
b5tgs;};}",
					"server dans{struct dans1{int b34twre;date bsdfsadf;date bsdfsadf2;};}",
					// Multiple w/ init
					"server dans{struct dans1{int b34twre=12345;boolean b5tgs=false;int
b34twre2=12345;};}",
					"server dans{struct dans1{int b34twre=12345;real b34rewf=454.054;};};}",
					// Mixed some init some not.
					"server dans{struct dans1{real b34rewf=454.054; boolean hey=false;string s; date d;};}",

		};
		testBatch(validBasix,true);
	}
	public void testStructInstances()
	{
		String validBasix[] = {
		"server dans{struct cat{string name=\"NA\";}; cat mine; cat yours;}",
		};
		testBatch(validBasix,true);

		String invalidBasix[] = {
					"server dans{struct cat{string name=\"NA\";}; catsss mine;}",
					};
					testBatch(invalidBasix,false);
	}
	public void testStructPropertyAccess()
	{
		String validBasix[] = {
		"server dans{struct cat{string name=\"NA\";}; cat mine; string cName = mine.name;}",
		};
		testBatch(validBasix,true);

		String invalidBasix[] = {
					"server dans{struct cat{string name=\"NA\";}; cats mine; int age = mine.age;}",
					};
					testBatch(invalidBasix,false);
	}
	public void testPropertiesAndVariables()
	{
		String validBasix[] = {
					// Property previously defined as variable
					"server dans{int i=12345; struct foo{int i=90;};};}",
					"server dans{int i=12345; struct foo{int i;};};}",

					// Property initalized to a variable
					"server dans{int i=12345; struct foo{int jol=i;};};}",
					// TODO what happens in above if foo has property i? Which value?

					//TODO later when semantics have it, try a function.

		};
		testBatch(validBasix,true);
```

```
	}
	/**
	 * Test some random structs.
	 */
	public void testRandomStructs()
	{
		Random r = new Random();
		int next = 0;
		StringBuffer s = null;
		for (int j=0; j < m_maxRandomToRun; ++j)
		{
			next = Math.abs(r.nextInt()) % m_maxDeclarations;
			s = new StringBuffer("server "+getRandomIdentifier()+" { struct "+ getRandomIdentifier()+ "{");

			for (int i=0; i < next; ++i)
			{
				s.append(declHelper.getRandomDecl(r));
			}
			s.append("\n};}");
			//NOPELEnv.flushErrors();
			NOPELErrorHandler.clearAllErrors();
			NOPELEnv.getSymbolTable().clearAllSymbols();
			basix(s.toString(),true);
		}
	}
	/**
	 * Test out struct access
	 *
	 */
	public void testStructAccess()
	{
		/*
		String validBasix[] = {		"server dans{int i= boring.id;}",
		};
		testBatch(validBasix,true);
		*/
	}
}
////////////////////////////////////////
////////////////////////////////////////
//TreeParserMain.java
//~~//////////////////////////////////////
//////////////////////////////////////////
package com.school.nopel.tests.semantics;

import java.util.Enumeration;
import java.util.Iterator;
import java.util.logging.Formatter;
import java.util.logging.Level;
import java.util.logging.LogManager;
import java.util.logging.LogRecord;
import java.util.logging.Logger;
import java.util.logging.SimpleFormatter;

import javax.swing.JFrame;

import com.school.nopel.NOPELASTFactory;
import com.school.nopel.NOPELLexer;
import com.school.nopel.NOPELParser;
import com.school.nopel.NOPELTreeParser;
import com.school.nopel.compiler.NOPELErrorHandler;

import antlr.collections.AST;
import antlr.debug.misc.ASTFrame;

public class TreeParserMain {
	static {
		initializeVerbosity(Level.ALL);
	}
   public static void main(String[] args) throws Exception
```

```
    {
        //NOPELLexer lexer = new NOPELLexer(System.in);
        NOPELLexer lexer = new NOPELLexer(System.in);
       NOPELParser parser = new NOPELParser(lexer);
       parser.setASTFactory(new NOPELASTFactory());
       parser.server();
       AST t = parser.getAST();
       NOPELTreeParser tp = new NOPELTreeParser();
       tp.server(t);

       if (!NOPELErrorHandler.isSemanticsCool())
       {
        System.out.println("Semantic errors:");
        Iterator it = NOPELErrorHandler.getSemanticErrors().iterator();
        while (it.hasNext())
        {
                        System.out.println(it.next());
        }
        return;
       }
       ASTFrame frame = new ASTFrame("AST JTree Example",t);
                        JFrame.setDefaultLookAndFeelDecorated(true);
                        frame.setVisible(true);
    }
    /**
         * Verbosity in the application is controlled via JavaLogger.
         *
         * @param _level
         */
        protected static void initializeVerbosity(Level _level) {
                        Logger.getLogger(".").setLevel(_level);
                        Formatter f = null;
                        f = new SimpleFormatter() {

                                public String format(LogRecord record) {
                                        if (null != record.getThrown()) {
                                                return record.getLevel() + ": " + record.getMessage()
                                                                + "\n" + record.getThrown().getMessage() + "\n";
                                        } else {
                                                return record.getLevel() + ": " + record.getMessage()
                                                                + "\n";
                                        }
                                }
                        };
                        Enumeration names = LogManager.getLogManager().getLoggerNames();
                        Logger l;
                        while (names.hasMoreElements()) {
                                l = ((Logger) (LogManager.getLogManager().getLogger(names
                                                .nextElement().toString())));
                                l.setLevel(_level);
                                if (l.getHandlers().length > 0)
                                        l.getHandlers()[0].setFormatter(f);
                        }
        }
}
```