CALIFORNIA STATE UNIVERSITY, Long Beach

Computer Science Department

CECS 543

Term project

“SIL”

Presented by:

Hashlamoun,Younes

Herman, Donald

Naumov,Mikhail

Instructor: Dr. Michael Hoffman

Fall 2010

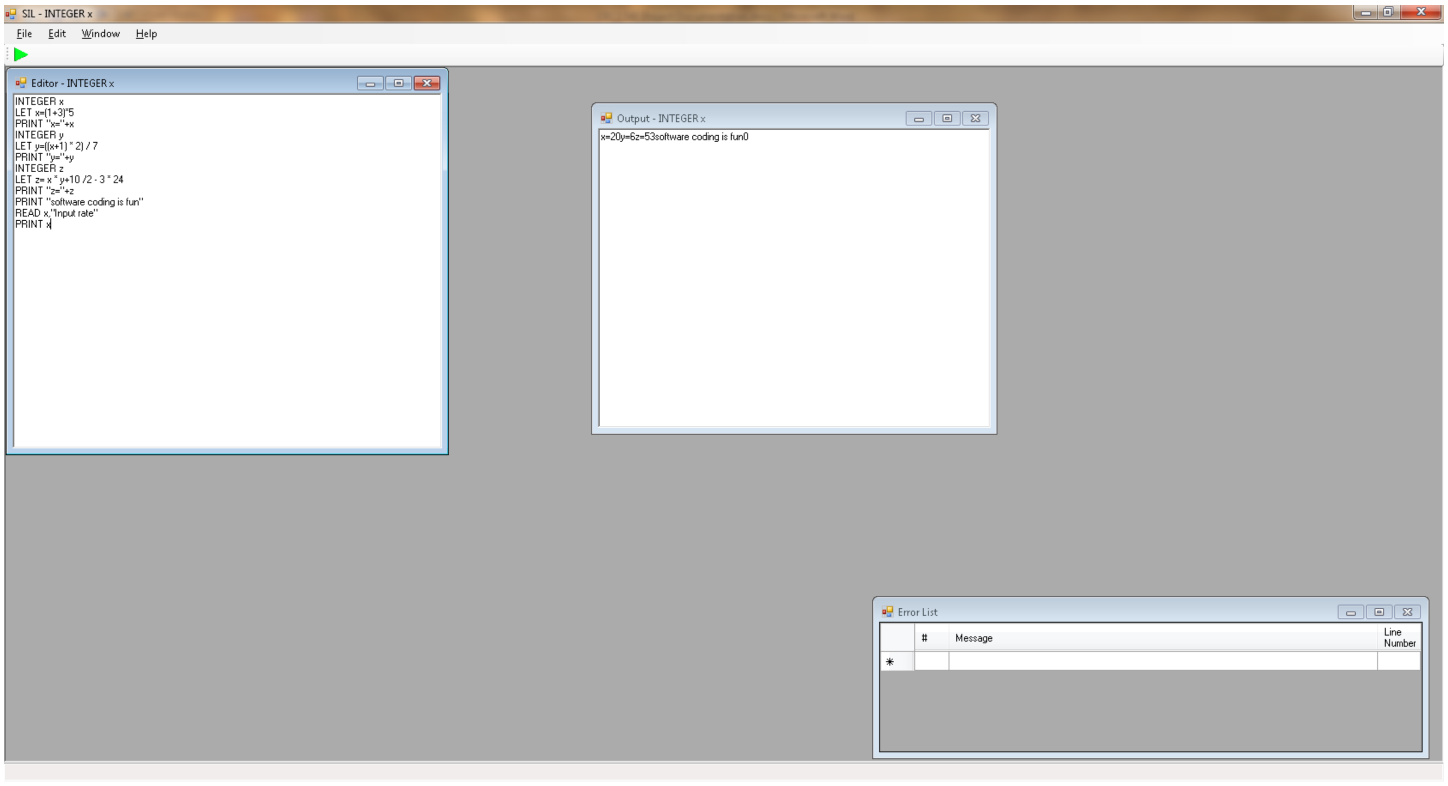
TABLE OF CONTENTS

List of features

Our project's features are very much fulfilling all the assignment's requirements. Our application is unique in many features also, like we do parse the input file by reading the whole text then verify it's syntax, if it passes with no errors, then we compile it or run it and send the expected output to the output screen.

Bellow is the list of features our application does have:

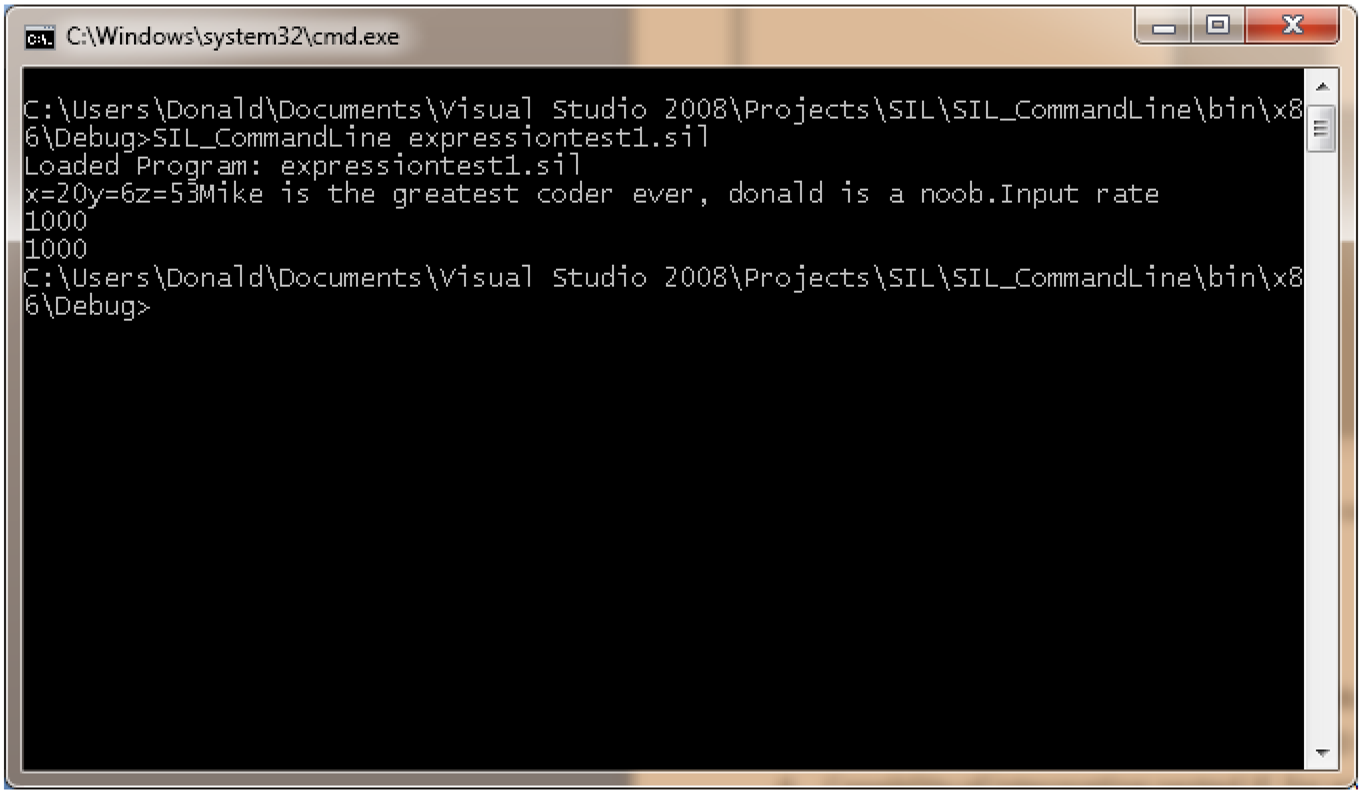
1. Two user interfaces, GUI we've got it as required, with main menu bar, status bar, input text window it can be multiple windows; one window for each file, output window and errors window.



```
INTEGER x
LET x=(1+3)*5
PRINT "x="+x
INTEGER y
LET y=((x+1) * 2) / 7
PRINT "y="+y
INTEGER z
LET z= x * y+10 /2 - 3 * 24
PRINT "z="+z
PRINT "software coding is fun"
READ x,"Input rate"
PRINT x
```

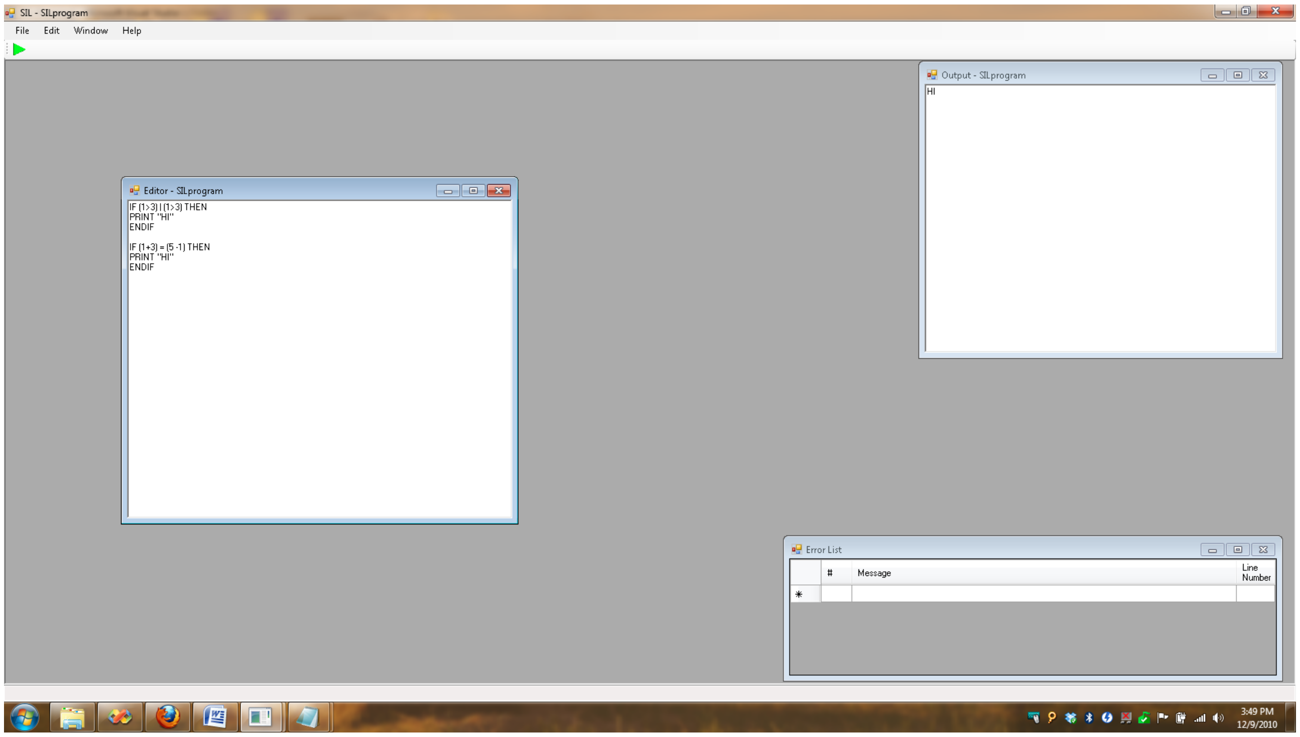
```
x=20y=6z=53software coding is fun0
```

2. Command prompt interface where you can pass the input file to be parsed as parameter argument to the application.



```
C:\Users\Donald\Documents\Visual Studio 2008\Projects\SIL\SIL_CommandLine\bin\x8
6\Debug>SIL_CommandLine expressiontest1.sil
Loaded Program: expressiontest1.sil
x=20y=6z=53Mike is the greatest coder ever, donald is a noob.Input rate
1000
1000
C:\Users\Donald\Documents\Visual Studio 2008\Projects\SIL\SIL_CommandLine\bin\x8
6\Debug>
```

3. Capability of interpreting all SIL keywords determined at the assignment, INTEGER, BEGIN, END, IF THEN, LET, FOR , WHILE, FUNCTION AND OPERATORS.
    a. IF

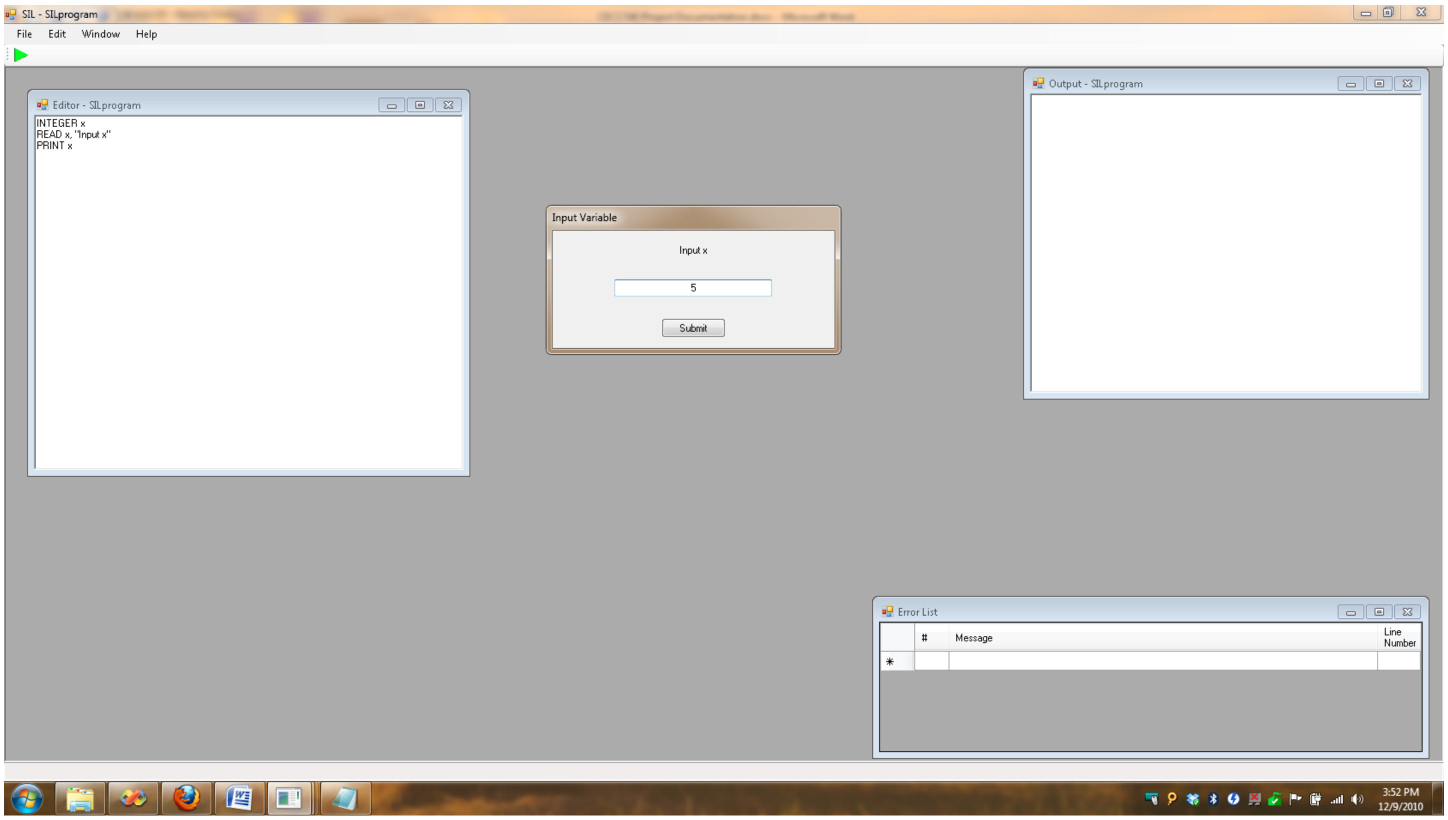


```
IF (1>3) | (1>3) THEN
PRINT "HI"
ENDIF

IF (1+3) = (5 -1) THEN
PRINT "HI"
ENDIF
```

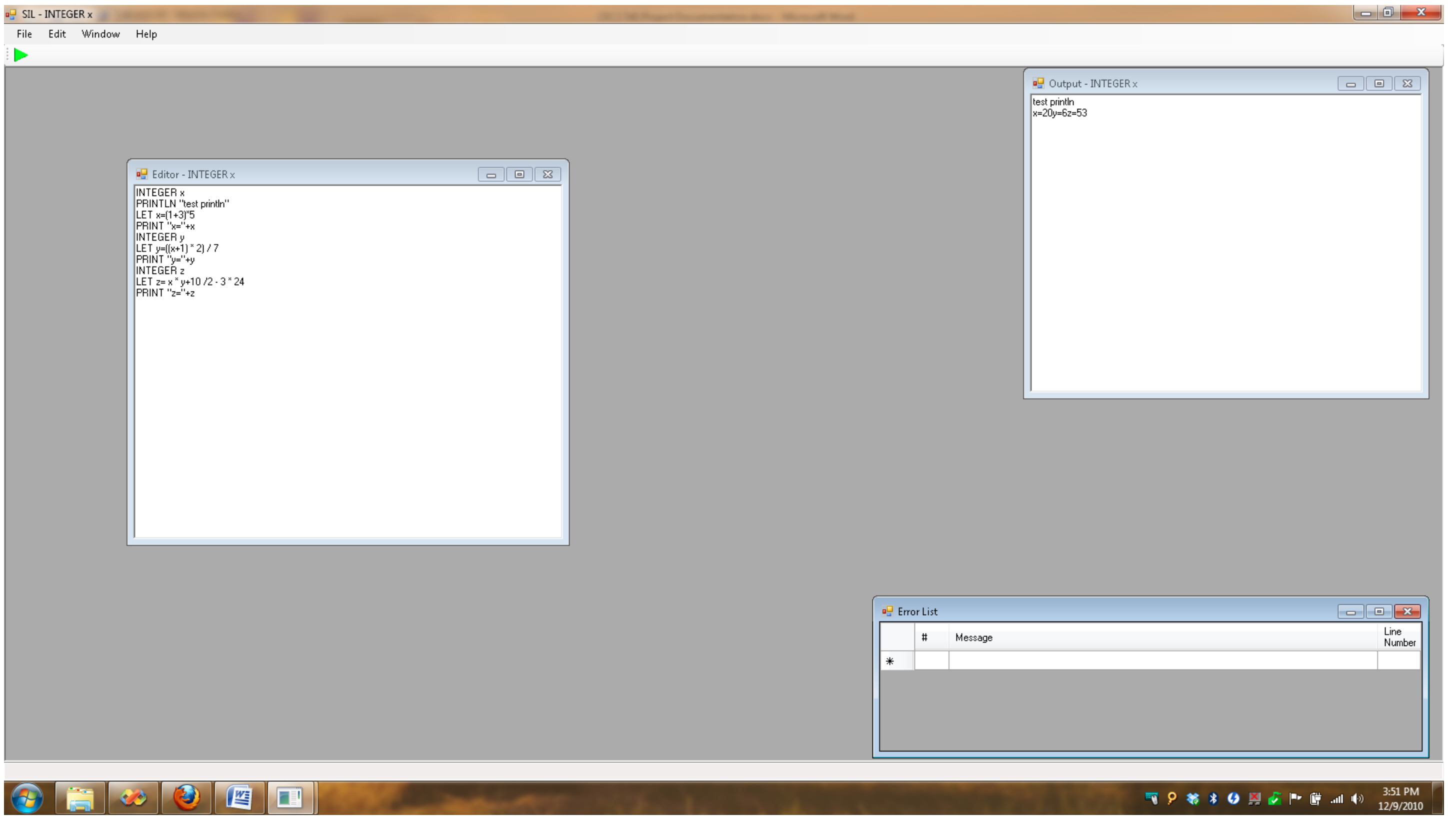
```
HI
```

b. Read



```
INTEGER x
READ x, "Input x"
PRINT x
```

4. Capability of interpreting nested if, for and while
5. Capability of evaluating mathematical expressions at any case, if used with let or print and at any level of complexity.



```
INTEGER x
PRINTLN "test println"
LET x=(1+3)*5
PRINT "x="+x
INTEGER y
LET y=((x+1) * 2) / 7
PRINT "y="+y
INTEGER z
LET z= x * y+10 /2 - 3 * 24
PRINT "z="+z
```

```
test println
x=20y=6z=53
```

6. Capability of detecting and reporting syntax errors with line number and nature of the error before proceeding to execution.
7. Capability of detecting and reporting run time errors.

Extra Credit Features

8. We have also implemented an array class that can hold INTEGER variables.

You need to declare array like this first:

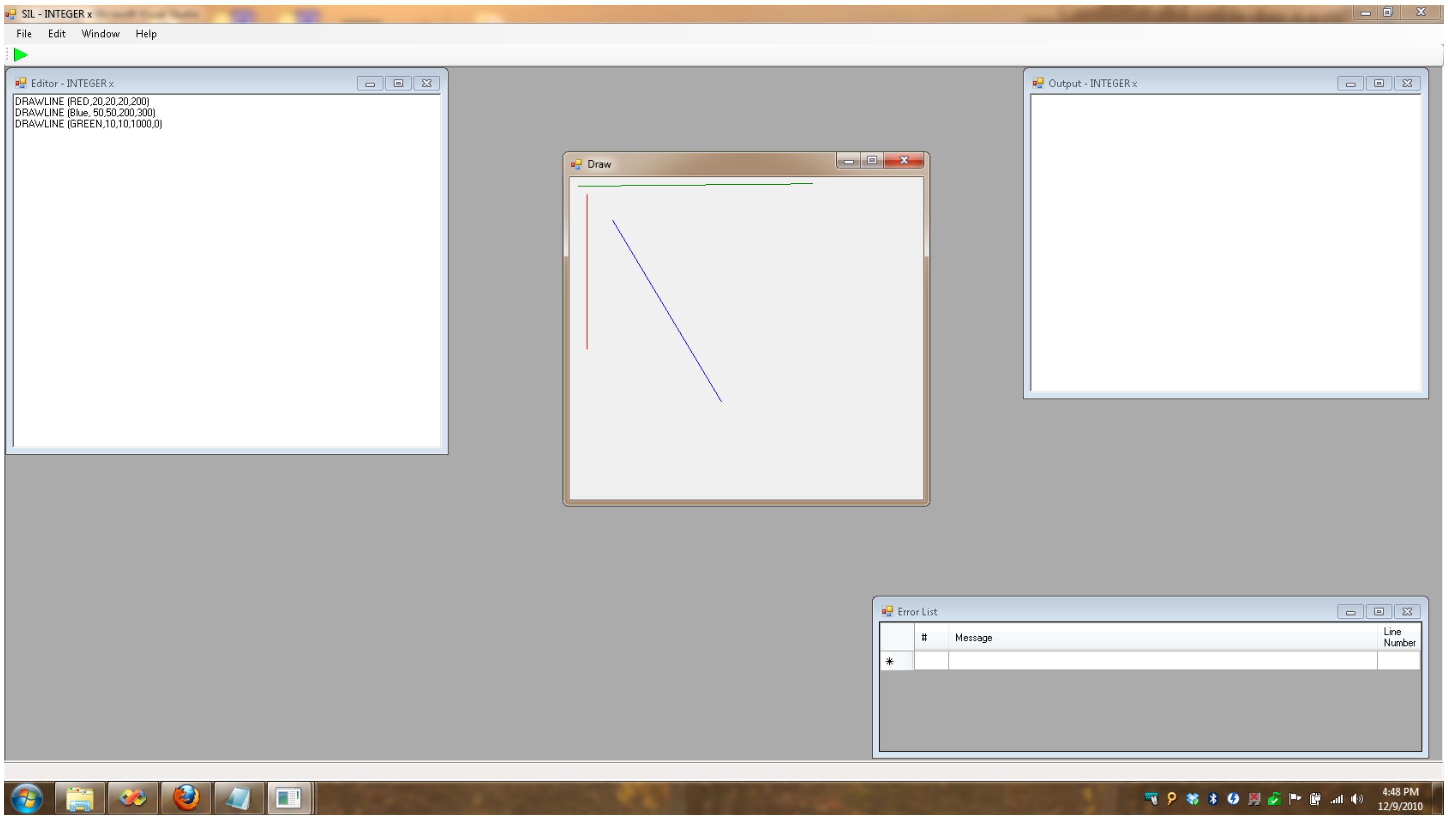
ARRAY myArray[10]

ARRAY myArray1[8192]
INTEGER a

After you declare an array you can set any of its elements:
//asign to array in a loop
FOR i = 0, i < 10, i = i+1
LET myArray[i] = 100 - i
PRINT arr2[i]
NEXT

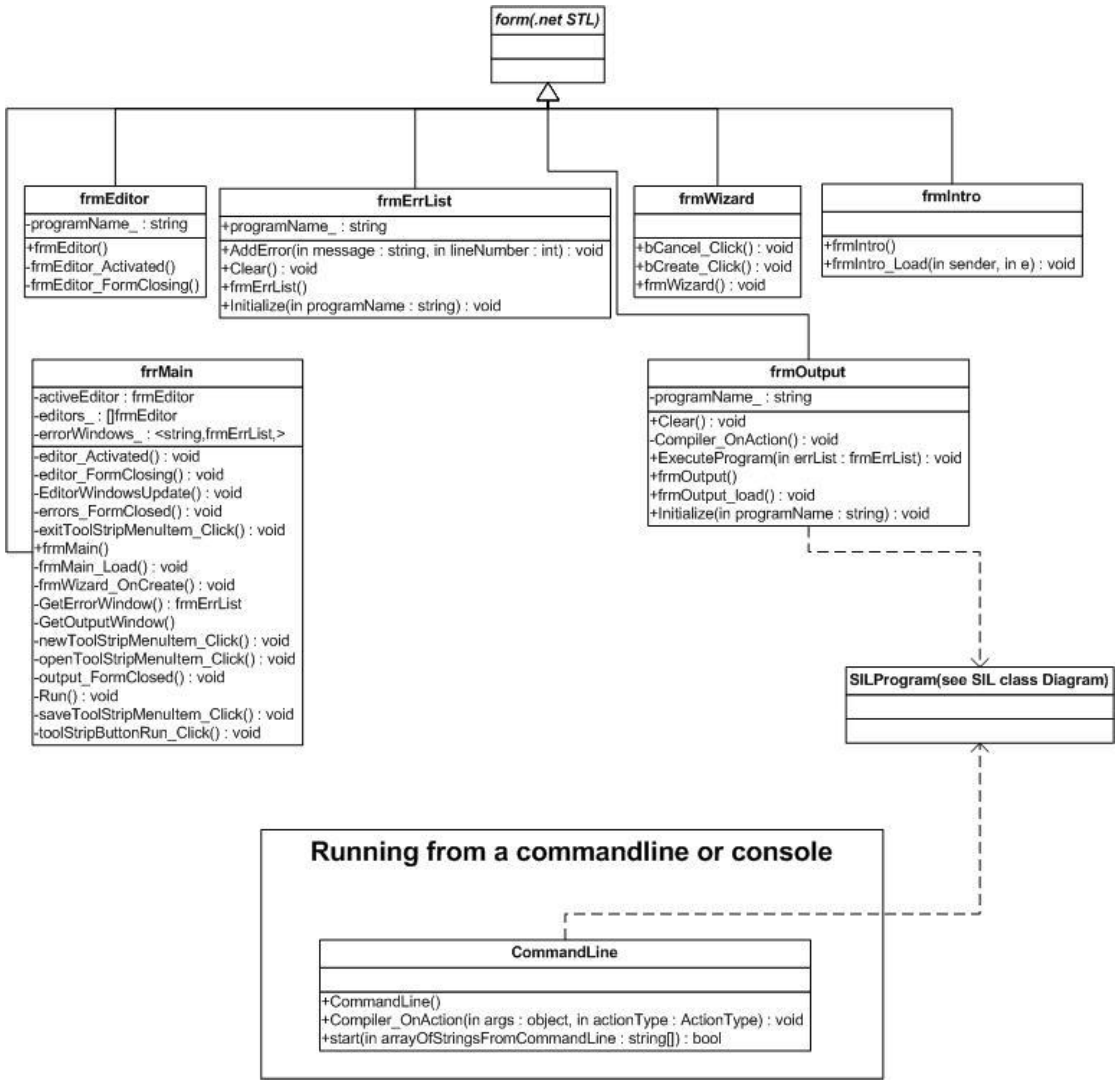
Also is that not only arrays can be used as other variables in equations, but you can pass equations as array indexes and even nest arrays one into another!
//array with a function
LET arr2[i] = arr1[i*10] * arr1[i*10]
//nested array
LET arr3[i] = arr2[arr1[i]]
//nested array with a function
LET arr3[i] = arr2[(arr1[i] - 1) * 2]

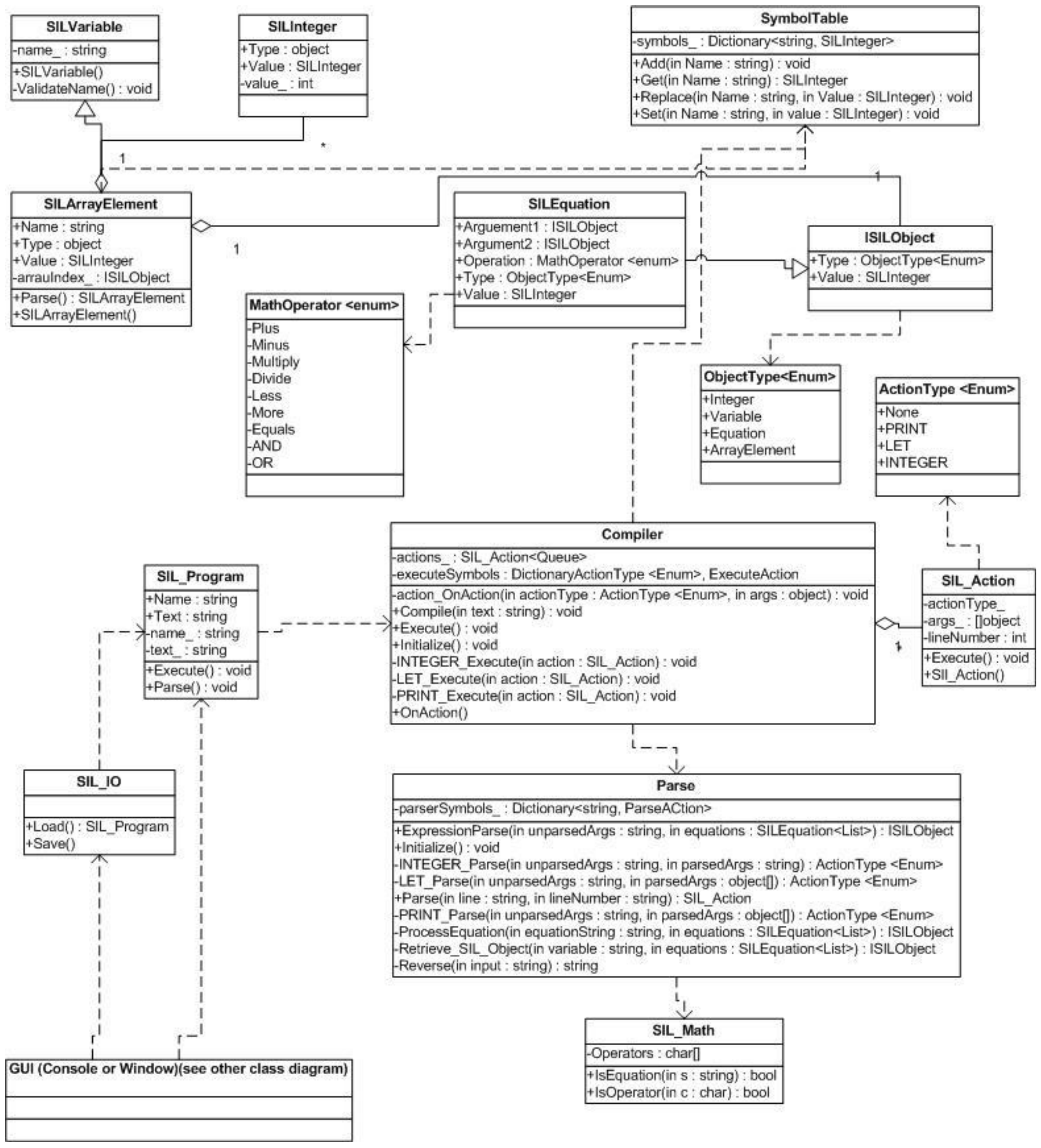
9. Graphics: The DRAWLINE method will use five parameters, the color of the line. It's starting X,Y locations and it's ending X,Y locations.



```
DRAWLINE (RED,20,20,20,200)
DRAWLINE (Blue, 50,50,200,300)
DRAWLINE (GREEN,10,10,1000,0)
```

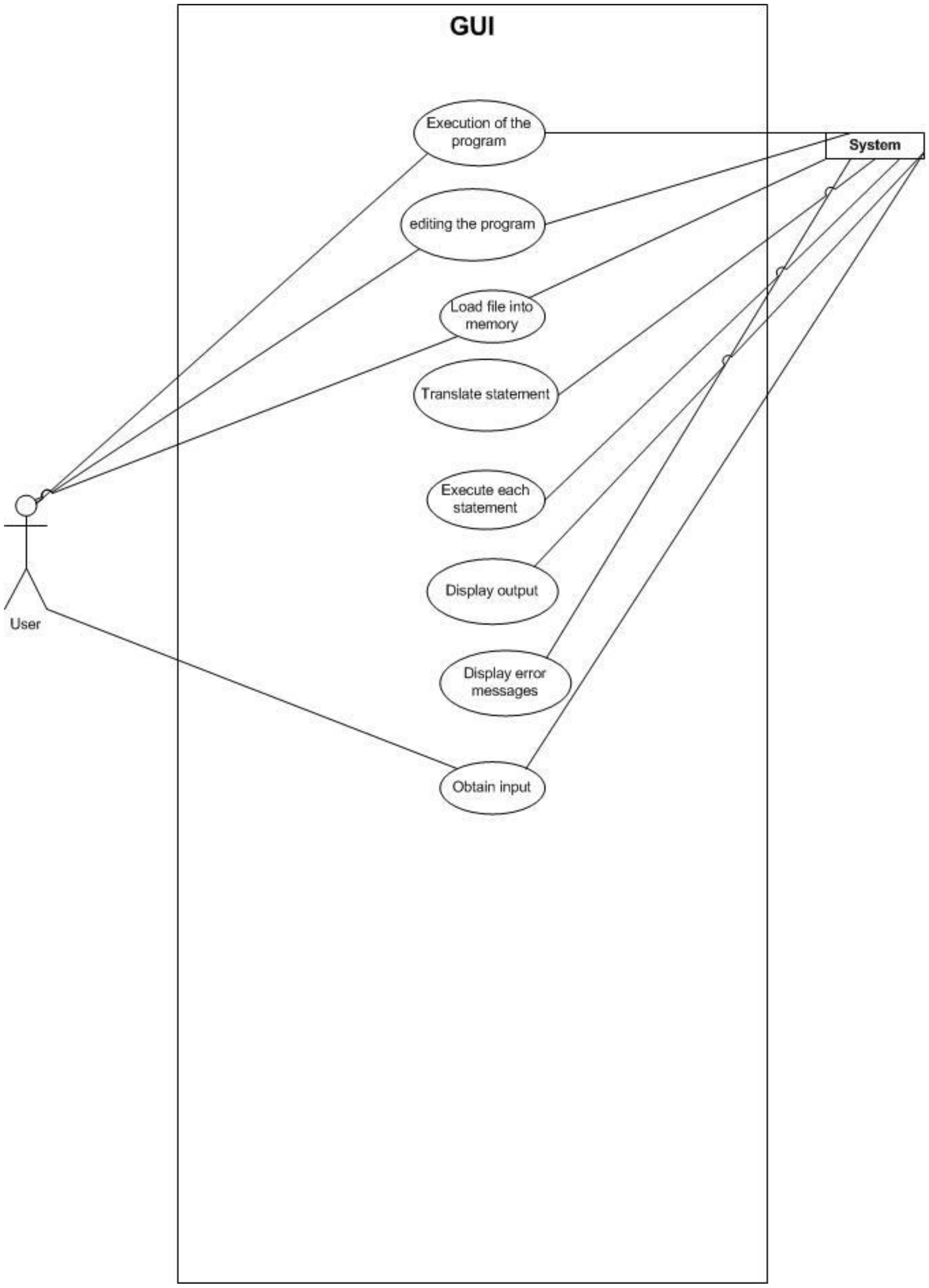
Class Diagram


form(.net STL)
frmEditor
-programName_ : string
+frmEditor()
-frmEditor_Activated()
-frmEditor_FormClosing()
frmErrList
+programName_ : string
+AddError(in message : string, in lineNumber : int) : void
+Clear() : void
+frmErrList()
+Initialize(in programName : string) : void
frmWizard
+bCancel_Click() : void
+bCreate_Click() : void
+frmWizard() : void
frmIntro
+frmIntro()
+frmIntro_Load(in sender, in e) : void
frrMain
-activeEditor : frmEditor
-editors_ : []frmEditor
-errorWindows_ : <string,frmErrList,>
-editor_Activated() : void
-editor_FormClosing() : void
-EditorWindowsUpdate() : void
-errors_FormClosed() : void
-exitToolStripMenuItem_Click() : void
+frmMain()
-frmMain_Load() : void
-frmWizard_OnCreate() : void
-GetErrorWindow() : frmErrList
-GetOutputWindow()
-newToolStripMenuItem_Click() : void
-openToolStripMenuItem_Click() : void
-output_FormClosed() : void
-Run() : void
-saveToolStripMenuItem_Click() : void
-toolStripButtonRun_Click() : void
frmOutput
-programName_ : string
+Clear() : void
-Compiler_OnAction() : void
+ExecuteProgram(in errList : frmErrList) : void
+frmOutput()
+frmOutput_load() : void
+Initialize(in programName : string) : void
SILProgram(see SIL class Diagram)
Running from a commandline or console
CommandLine
+CommandLine()
+Compiler_OnAction(in args : object, in actionType : ActionType) : void
+start(in arrayOfStringsFromCommandLine : string[]) : bool

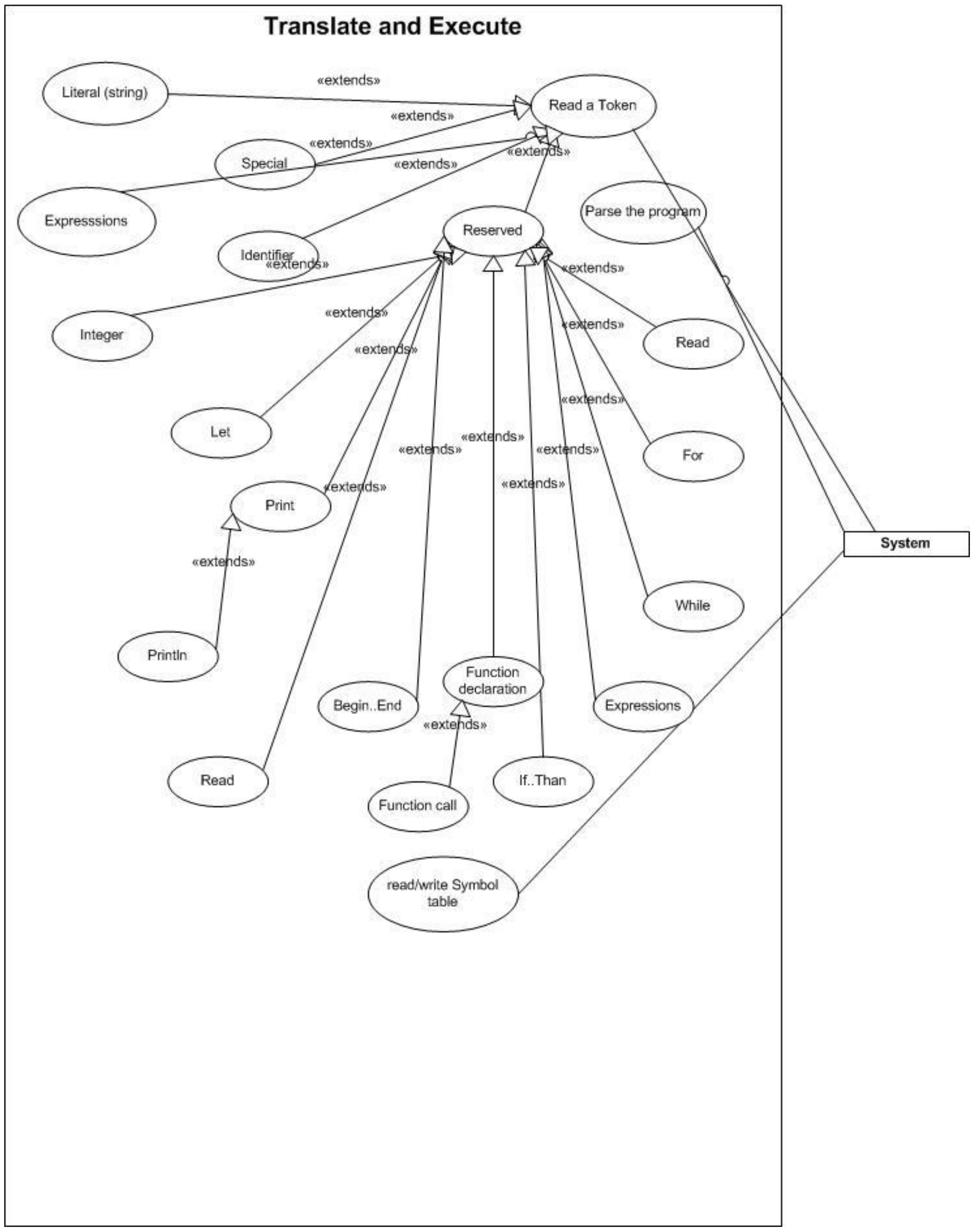

SILVariable
-name_ : string
+SILVariable()
-ValidateName() : void
SILInteger
+Type : object
+Value : SILInteger
-value_ : int
SymbolTable
-symbols_ : Dictionary<string, SILInteger>
+Add(in Name : string) : void
+Get(in Name : string) : SILInteger
+Replace(in Name : string, in Value : SILInteger) : void
+Set(in Name : string, in value : SILInteger) : void
1
*
1
1
SILArrayElement
+Name : string
+Type : object
+Value : SILInteger
-arrauIndex_ : ISILObject
+Parse() : SILArrayElement
+SILArrayElement()
SILEquation
+Arguement1 : ISILObject
+Argument2 : ISILObject
+Operation : MathOperator <enum>
+Type : ObjectType<Enum>
+Value : SILInteger
ISILObject
+Type : ObjectType<Enum>
+Value : SILInteger
MathOperator <enum>
-Plus
-Minus
-Multiply
-Divide
-Less
-More
-Equals
-AND
-OR
ObjectType<Enum>
+Integer
+Variable
+Equation
+ArrayElement
ActionType <Enum>
+None
+PRINT
+LET
+INTEGER
Compiler
-actions_ : SIL_Action<Queue>
-executeSymbols : DictionaryActionType <Enum>, ExecuteAction
-action_OnAction(in actionType : ActionType <Enum>, in args : object) : void
+Compile(in text : string) : void
+Execute() : void
+Initialize() : void
-INTEGER_Execute(in action : SIL_Action) : void
-LET_Execute(in action : SIL_Action) : void
-PRINT_Execute(in action : SIL_Action) : void
+OnAction()
SIL_Program
+Name : string
+Text : string
-name_ : string
-text_ : string
+Execute() : void
+Parse() : void
SIL_Action
-actionType_
-args_ : []object
-lineNumber : int
+Execute() : void
+Sil_Action()
1
SIL_IO
+Load() : SIL_Program
+Save()
Parse
-parserSymbols_ : Dictionary<string, ParseACtion>
+ExpressionParse(in unparsedArgs : string, in equations : SILEquation<List>) : ISILObject
+Initialize() : void
-INTEGER_Parse(in unparsedArgs : string, in parsedArgs : string) : ActionType <Enum>
-LET_Parse(in unparsedArgs : string, in parsedArgs : object[]) : ActionType <Enum>
+Parse(in line : string, in lineNumber : string) : SIL_Action
-PRINT_Parse(in unparsedArgs : string, in parsedArgs : object[]) : ActionType <Enum>
-ProcessEquation(in equationString : string, in equations : SILEquation<List>) : ISILObject
-Retrieve_SIL_Object(in variable : string, in equations : SILEquation<List>) : ISILObject
-Reverse(in input : string) : string
SIL_Math
-Operators : char[]
+IsEquation(in s : string) : bool
+IsOperator(in c : char) : bool
GUI (Console or Window)(see other class diagram)

I. Use cases

Use case GUI



Use case translate execute



II. Use Case Documentation

Begin…End Statement

Primary actor: computer

Goal in context: This statement allows a series of statements to be executed as one.

Preconditions: computer has parsed the code

Post Condition: End Token has been reached

Trigger: computer encounters Begin token and the being event has fired.

Scenario:

	1. Computer executes the remaining lines of statements
	2.

Exceptions:

1.

Display error messages

Primary actor: Computer

Goal in context: A syntax or run time error is displayed to the user.

Preconditions: program has been successfully parsed and is executing.

Post Condition: error window or command line has been updated

Trigger: A syntax or run time error has occurred.

Scenario:

	1. Error message is passed as a string literal to the output windows of the correct user interface.
	2. If user interface is the GUI than the error message window's text variable is updated

	3. If user interface is the command line than the error message is written to the command line.

Exceptions:

1.

Display Output

Primary actor: computer

Goal in context: Display the output of the computer program

Preconditions: Program has been successfully parsed and is executing.

Post Condition: Message has been displayed.

Trigger: A token that displays a message has been encountered

Scenario:

	1. The command that display a message evaluates the string literal in question.
	2. The string literal that is displayed is passed the output windows
	3. If GUI is active than the output window's display is update with the new string literal.
	4. If command Line is active than the string literal is displayed on the command line.

Exceptions:

1.

editing the program

Primary actor: User

Goal in context: user can change the code in the program and save the file

Preconditions: file is open and displayed in the editor

Post Condition: file has been saved to disk.

Trigger: User clicks on open file.

Scenario:

1. User can add and delete characters in the editing window.	2. Computer updates the information displayed in the editing window
3. User selects file > save to save the edited file to disk	4. Computer saves the new information in the editing windows to disk in the file specified.

Exceptions:

1.

Execute each statement

Primary actor: computer

Goal in context: computer executes each statement in the code.

Preconditions: code has been checked for syntax errors

Post Condition: computer has encountered END statement in the main method.

Trigger: first line has been encountered.

Scenario:

	1. First line of code is encountered by computer and it is parsed.
	2. The token are checked and the proper methods are executed as needed.

Exceptions:

1.

Execution of the program

Primary actor: User

Goal in context: Running an Interpreted sim file.

Preconditions: sim file must be open

Post Condition: computer starting to execute the code.

Trigger: user selects run

Scenario:

1. User clicks on the run button.	2. Computer starts to translate the file.

Exceptions:

1. There is no file loaded; error message to shown please open a file.	

Expressions

Primary actor: computer

Goal in context: Constants, identifies and arithmetic operators are used to complete a single integer value.

Preconditions: Code has been successfully parsed.

Post Condition: Expression method has returned a single integer value.

Trigger: When certain REVERED words (LET, IF, while, for) or a SPECIAL character is encounter by the computer.

Scenario:

	1. When a special character is found the following token in the lineup until a terminator token (then, end of line, comma) are used as parameters for the expression method.
	2. Leading token is compared against a switch statement of special characters.
	3. Second Token is computed by Solution integer value according to the special character. (Example Token is 1, special character is +, solution integer is 5, new solution integer is 1 + 5 = 6.)
	4. If Token is an identifier when obtain value from Symbol Table.
	5. Remaining tokens and Solution integer is passed to the expression method recursively.
	6. This recursion continues as long as tokens remain in the container of tokens.

	7. The solution integer is returned when all tokens are used.

Exceptions:

1.

2. Token doesn't match any items in the switch statement Synax error message is displayed "Expression is invalid Line #X."

3. If SPECIAL symbol is divide (/) and the token is zero. Computer returns a run time error "Divide by zero Line#".

FOR Statement

Primary actor: computer

Goal in context: The FOR Statement allows execution of the following statement block (between begin … end blocks) until a counter is exceeded.

Preconditions: Program has been successfully parsed.

Post Condition: Block statement has been executed the specified number of times.

Trigger: The computer encounters the FOR token and FOR statement event is triggers.

Scenario:

	1. The tokens following the FOR token until the END state are passed to the FOR method in a container of tokens.
	2. The first token or series of tokens are used to set the initial value for the counter until the DO toke is encountered.
	3. The next sets of tokens are used to set the terminator for the counter until the DO token is encountered.
	4. The statements following the BEGIN token is executed as normal.
	5. Once the END token is encountered the counter is incremented by one and compared to the terminator. If less than the terminator than goto step #4.
	6. Once terminator has been reach the computer

	will skip the following begin end block.

Exceptions:

2. Token or series must be a real number else syntax error is encountered and computer displays "Error real number must be used line # X."

3. Token or series must be a real number else syntax error is encountered and computer displays "Error real number must be used line # X."

Function declaration

Primary actor: computer

Goal in context: A function name is encountered that has been added to the hash table of reserved word.

Preconditions: The computer has successfully parsed the code and the main part of the application program is running.

Post Condition: function end statement has been encountered.

Trigger: computer encounters a function name and the function event is triggered.

Scenario:

	1. Parameters are passed to the function event
	2. The line of code in the function are executed as normal

Function declaration

Primary actor: computer

Goal in context: A function name is added to the harsh table of Reserved word and the statements in between the begin … end block are added to the event manager under the function name.

Preconditions: The computer has successfully parsed the code.

Post Condition: function statement have been added to the event manager as an event.

Trigger: computer encounters a function token and the function event is triggered.

Scenario:

	3. Tokens following the "function" token are passed to the function event ending with the

	END token.
	4. The first token is the identifier and is added to the event manager hash table
	5. The remaining tokens are stored in a function object as a container of tokens.

Exceptions:

1. If a “begin” or “function” token is encountered before the “end” token than the program terminates with a syntax error ” must enclose block with begin and end line #”
2. If hash table already contains the identifier than the computer terminates and then displays the syntax error message “function has already been declared.”
3. Tokens must be checked for list of parameters
   a. The first token must be a opening parentheses
   b.

Identifier

Primary actor: computer

Goal in context: Identifier identifier value can obtain in constant time from Hash Table (symbol table).

Preconditions: code is being interpreted and executed.

Post Condition: Identifier value has been returned.

Trigger: interpreter has encountered identifier token and the identifier method event has fired.

Scenario:

	1. Identifier from symbol table is looked up using hash code algorithm.
	2. Identifier value is returned

Exceptions:

1. If identifier has not been declared than program stops and a syntax error is displayed “Identifier has not been declared line # X”

IF…Then Statement

Primary actor: computer

Goal in context: Allows SIL code to conditionally execute a statement.

Preconditions: code is parsed successfully

Post Condition: conditional statement is executed or skipped.

Trigger: Token containing the reserved word " IF" is encountered.

Scenario:

	1. When IF token is encountered event is triggered and the IF..Then method is called.
	2. All token are assume to be a valid relational expression before the then statement
	3. Relational expression event is triggered and the expression method is called.
	4. The Relational expression method returns a Boolean value.
	5. If Boolean value from step #4 is true than the statement line after the THEN token is executed
	6. If Boolean value from step #4 is true than the statement line after the THEN token is not executed.

Exceptions:

1.

2. If THEN token is missing than the computer displays a syntax error "incomplete IF THEN Statement" and line #

3. If Relational expression divides by zero than a run time error displayed "Divide by Zero" Line #.

Let Statement

Primary actor: computer

Goal in context: Let Statement will assign a value to a variable using an expression

Preconditions: Interpreter encounters token is the Let reserved word.

Post Condition: Symbol Table variable value is updated.

Trigger: Let reserved word triggers event to fire.

Scenario:

	1. All tokens in the Let line are passed to the Let method.
	2. The first token is assumed to be a variable in Symbol Table.
	3. The expression is evaluated using the expression methods.
	4. The variable is updated to the new value returned by the expression method.

Exceptions:

2. If the Symbol Table doesn't contain the variable a Syntax error is displayed and the program stops executing.

3. If the Expression divides by zero than a run time error is displayed and the program stops running.

Literal

Primary actor: computer

Goal in context: The handling of Literals by the interpreter.

Preconditions: code is being interpreted and executed.

Post Condition: Literal has been displayed on screen.

Trigger: A double quote token has been encountered.

Scenario:

	1. Token following the double quote are treated as the same Literal
	2. Second double quote token is encountered and treated as the end of the Literal.

Exceptions:

2. If no second double quote is encountered before end of line than program terminates with Syntax error "Literal must be enclosed by quotes Line # x"

Load file into memory

Primary actor: user and system

Goal in context: File is selected by user and loaded into memory for editing and executing.

Preconditions: SIL GUI is loaded and running.

Post Condition: file is open and displayed on the GUI screen

Trigger: User selects file > open from GUI menu

Scenario:

1. User selects *.sim file and clicks open	2. Computer displays file in the editing window

Exceptions:

1. User selects a non sim file; file should be opened as if it was a sim file.	2.

Obtain Input

Primary actor: User and Computer

Goal in context: Obtain input from the user in order to update an Identifier's value

Preconditions: Code has been parsed and is running.

Post Condition: User has finished inputting data.

Trigger: An input method has been triggered.

Scenario:

	1. Computer display curser on the output screen according to the User interface
	2. If GUI is user interface than the output windows display the curser.
	3. If command line is user interface than the command line displays the curser.
4. User input value into the UI and a carrier return.	5. Computer takes value from user input and stores it in memory.
	6. The variable value is updated in the Symbol table.

Exceptions:

4. If user inputs a none real number than the program with terminate with a syntax error. “Input must be a real number line #.” Application programmer must check for correct input and let the user retry in order for program to not crash.

Parse the program

Primary actor: computer

Goal in context: SIL file will be parse line by line to collect and analyzed each token as needed.

Preconditions: SIL file is load and run.

Post Condition: Entire file is successfully parsed.

Trigger: User clicks on run

Scenario:

	1. File is open using STL file reader
	2. First line is parsed using space as a delimiter. Each token is stored in an index of an array of string.
	3. Each token is stored in an index of an array of strings.
	4. The first token is compared against a hash table of REVERED words.
	5. An event is created and the parameters, type and line number of the event are stored.
	6. Step 4 to 10 are repeated for each line number.

Exceptions:

	8. If token is not a REVERED word than exception is created.

Print Statement Reserved Word

Primary actor: computer

Goal in context: Execute print command

Preconditions: Code has been parsed and a reserved word has been found.

Post Condition: literal is displayed

Trigger: Computer finds reserve word and event is fired.

Scenario:

	1. Print method is executed and the rest of the tokens are passed to the method.
	2. If next token is an expression the expression event is fired.
	3. Expression is displayed
	4. If token is Literal it is displayed
	5. Step 2 to 4 are continued until all the token have been evaluated.

Exceptions:

2. If expression divides by zero a run time error is display "divide by zero Line #" X and the program terminates.

PRINTLN Statement

Primary actor: computer

Goal in context: All items expression and literals are displayed with a carrier return.

Preconditions: Code has been parsed

Post Condition: computer has finished displaying all the literals, expressions and displayed a new line

Trigger: PRINTLN token is encountered by the computer

Scenario:

	6. Print method is executed and the rest of the tokens are passed to the method.
	7. If next token is an expression the expression event is fired.
	8. Expression is displayed
	9. If token is Literal it is displayed

	10. Step 2 to 4 are continued until all the token have been evaluated.
	11. Carriage return and newline are displayed.

Exceptions:

3. If expression divides by zero a run time error is display “divide by zero Line #” X and the program terminates.

Read a Token

Primary actor: Computer

Goal in context: Compares parsed token and executes appropriate commands

Preconditions: Line of code has been parsed by compiler.

Post Condition: Token has been compared to list of tokens.

Trigger: new line of code is encountered.

Scenario:

	1. Parsed Token is compared against hash table of Tokens
	2. If token is found in table than appropriate commands are executed (see use cases that extends this use case)

Exceptions:

1. If token is missing than execution is displayed to user with command line #. Code is not executed.

Read Statement

Primary actor: Computer

Goal in context: This command allows user input to be saved in a variable.

Preconditions: Code has been parsed and code is running.

Post Condition: variable has been updated.

Trigger: A “READ” token has been encountered and the READ event has been triggered.

Scenario:

	1. User is prompted to input data
	2. User input is obtained
	3. Variable is updated with user input

Exceptions:

2. If user input is not a number than a run time error will be displayed. Ideally Application programmer must check for errors.

Read a Token (Reserved)

Primary actor: computer

Goal in context: actions that Interpreter takes when encountering a Reserved Token

Preconditions: Code has been Parsed by the computer

Post Condition: Computer has finished executing the line of code

Trigger: Token read is a "Reserved" Token

Scenario:

	1. Token is compared to hash table of reserved words
	2. If match is found event is triggered

Exceptions:

1. If reserved word is not found Token is assumed to be a Identifier

Special characters

Primary actor: computer

Goal in context: Special characters used in IF…Then , Let and expressions Commands

Preconditions: code is being interpreted and executed.

Post Condition:

Trigger: Special character token is encountered by the token

Scenario:

Exceptions:

1.

read/write symbol table

Primary actor: computer

Goal in context: add item to the symbol table and read and write to them

Preconditions: code is being interpreted and executed.

Post Condition: value is returned or symbol table is updated.

Trigger: INTEGER token is encountered by interpreter

Scenario:

	1. Token found by the interpreter after the integer statement are added to the symbol table
	2. If Token is not a reserved word it is assumed to be a INTEGER and the its value in the symbol table is return
	3. If token after integer is an assignment token than value assign will be added to the symbol table.

Exceptions:

1. If token is already found in the symbol table than Compile error is generated "identifier is already in the symbol table."
2. If value is NULL than a Compile error is generated "identifier has null have value assigned."
3. If value is not an integer value than a Compile error is generated "identifier has null have value **Use case:** Translate statement

Primary actor: computer

Goal in context: computer translates the statement of the program

Preconditions: user has clicked on run and has a file open

Post Condition: line has finished executing.

Trigger: user has click on run.

Scenario:

	1. Tokens encountered by the computer are analyzed
	2. Proper methods are executed according to the tokens encountered.

Exceptions:

1.

While Statement

Primary actor: computer

Goal in context: statement line below the while statement continually executed until while the expression is true

Preconditions: code has been parsed successfully.

Post Condition: expression is false and statement is skipped.

Trigger: computer encounters token that is a WHILE statement and event is triggered.

Scenario:

	1. All tokens following the while token and before the DO token are considered part of the relational expression. These token are passed as a container of strings to the relational expression method.
	2. If than relation expression is true than the statement below the while . do is executed. If condition is false do to step 4.
	3. Steps 1 and 2 are continued.
	4. Statement below while .. do is not executed.

Exceptions:

2. If relational expression divides by error than computer displays message. Run time error" divide by zero line #"

Function Call

Primary actor: computer

Goal in context: Executing a function using the values in the specified parameters.

Preconditions: code has been successfully parsed by the computer, and main body of programming is been executed.

Post Condition: Function END token has been encountered.

Trigger: Token of function Identifier is encountered and function event is triggered.

Scenario:

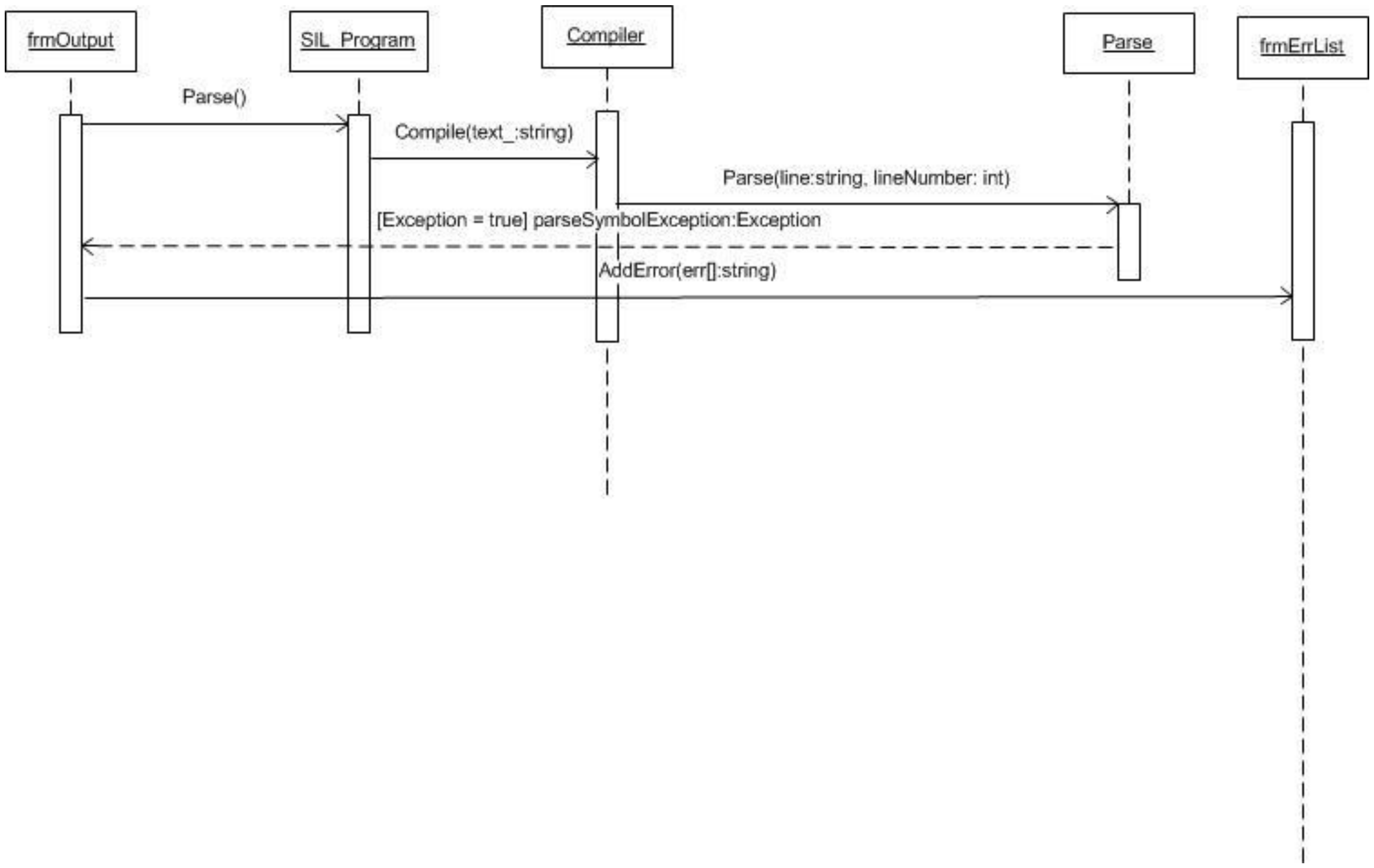
	1. Tokens starting from the open parenthesis to a closed parenthesis are passed to the event
	2. The numbers of tokens between the parenthesis are compared to the parameter list.
	3. A BEGIN token is encounter in the function.
	4. Each statement after the begin token is executed.
	5. When the return Token is encountered the function event terminates.
	6. An integer value is returned.

Exceptions:

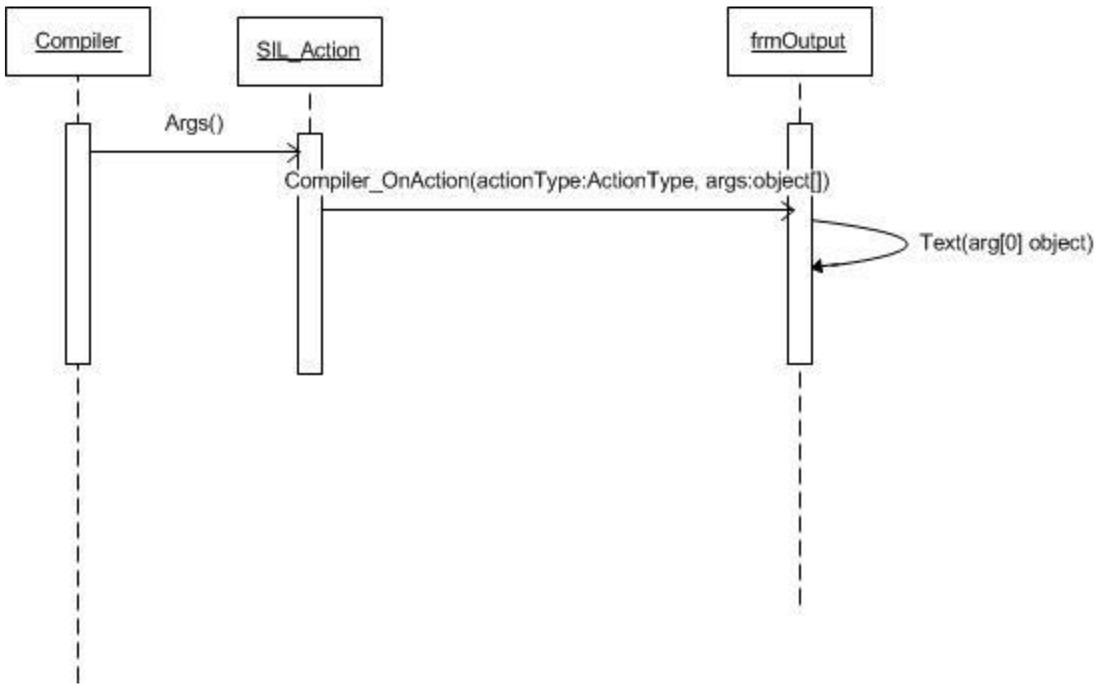
1. If the paraphrases are missing than an syntax error is displayed. "Missing parenthesis. Line # x"
2. If the number of token in the function call don't match the declaration than the computer stops and a syntax error is displayed. "Incorrect # of parameters. Line # x"
3. If a begin token is not encountered than the program terminates and a syntax error is displayed. "Missing begin token in function line # X"
5. If "end" token is encountered before "return" token program terminates and syntax error message is displayed "Missing return token in function line # X"
6. If none integer value is retuned example "string" than program terminates and syntax error message is displayed "Missing return return integer value in function line # X"

III. Sequence Diagrams

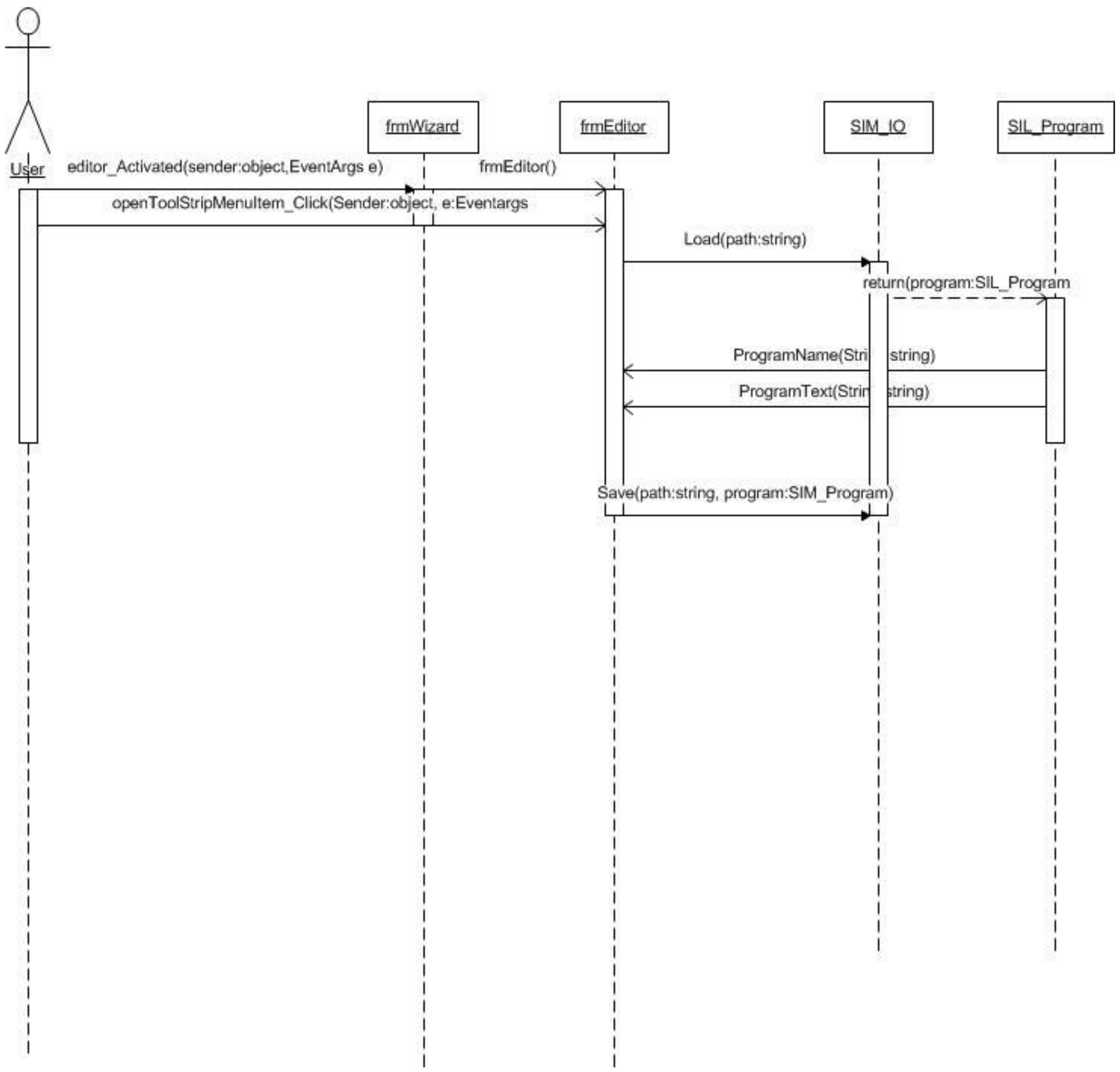
Display error message



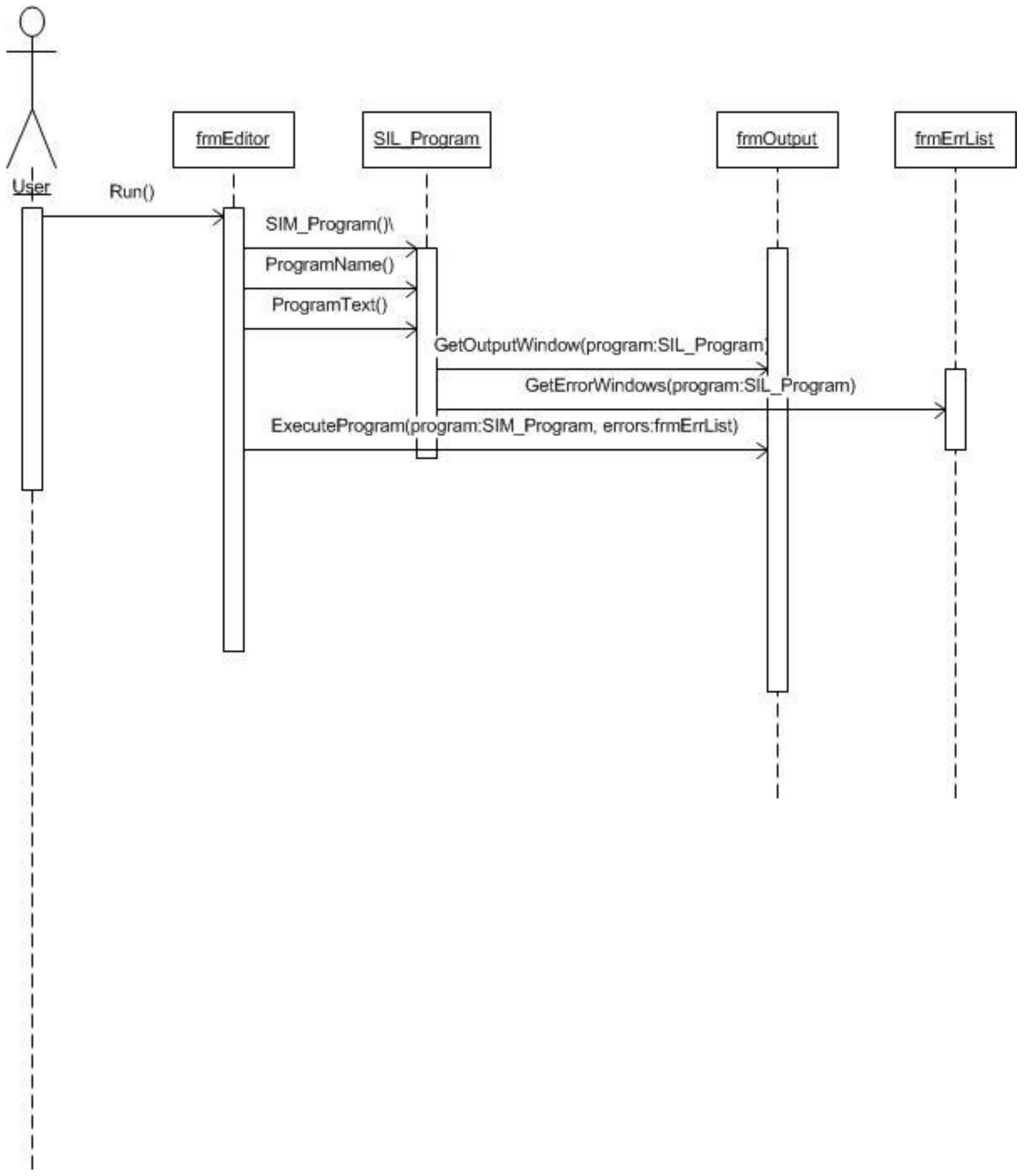
Display Output



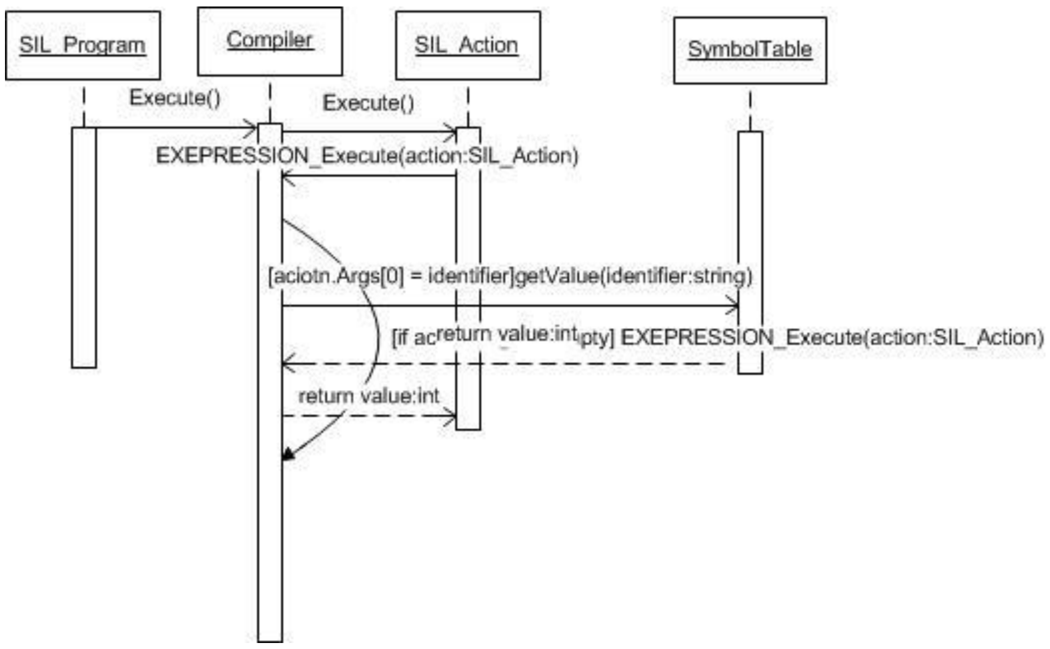
Editing the Program


User
frmWizard
frmEditor
SIM_IO
SIL_Program
editor_Activated(sender:object,EventArgs e)
frmEditor()
openToolStripMenuItem_Click(Sender:object, e:Eventargs
Load(path:string)
return(program:SIL_Program
ProgramName(Stri string)
ProgramText(Strir string)
Save(path:string, program:SIM_Program)

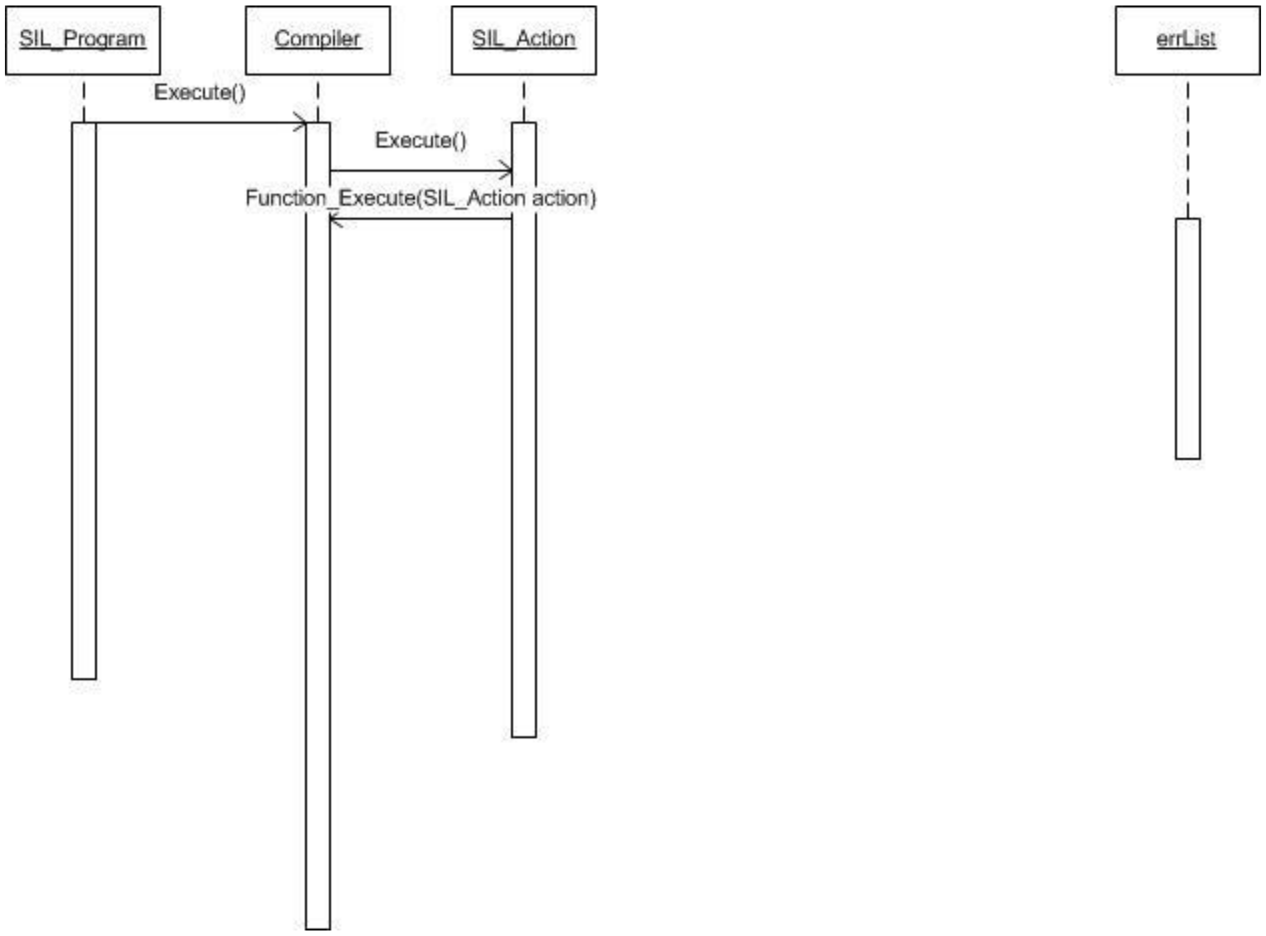
Execution of the Program


User
frmEditor
SIL_Program
frmOutput
frmErrList
Run()
SIM_Program()\
ProgramName()
ProgramText()
GetOutputWindow(program:SIL_Program)
GetErrorWindows(program:SIL_Program)
ExecuteProgram(program:SIM_Program, errors:frmErrList)

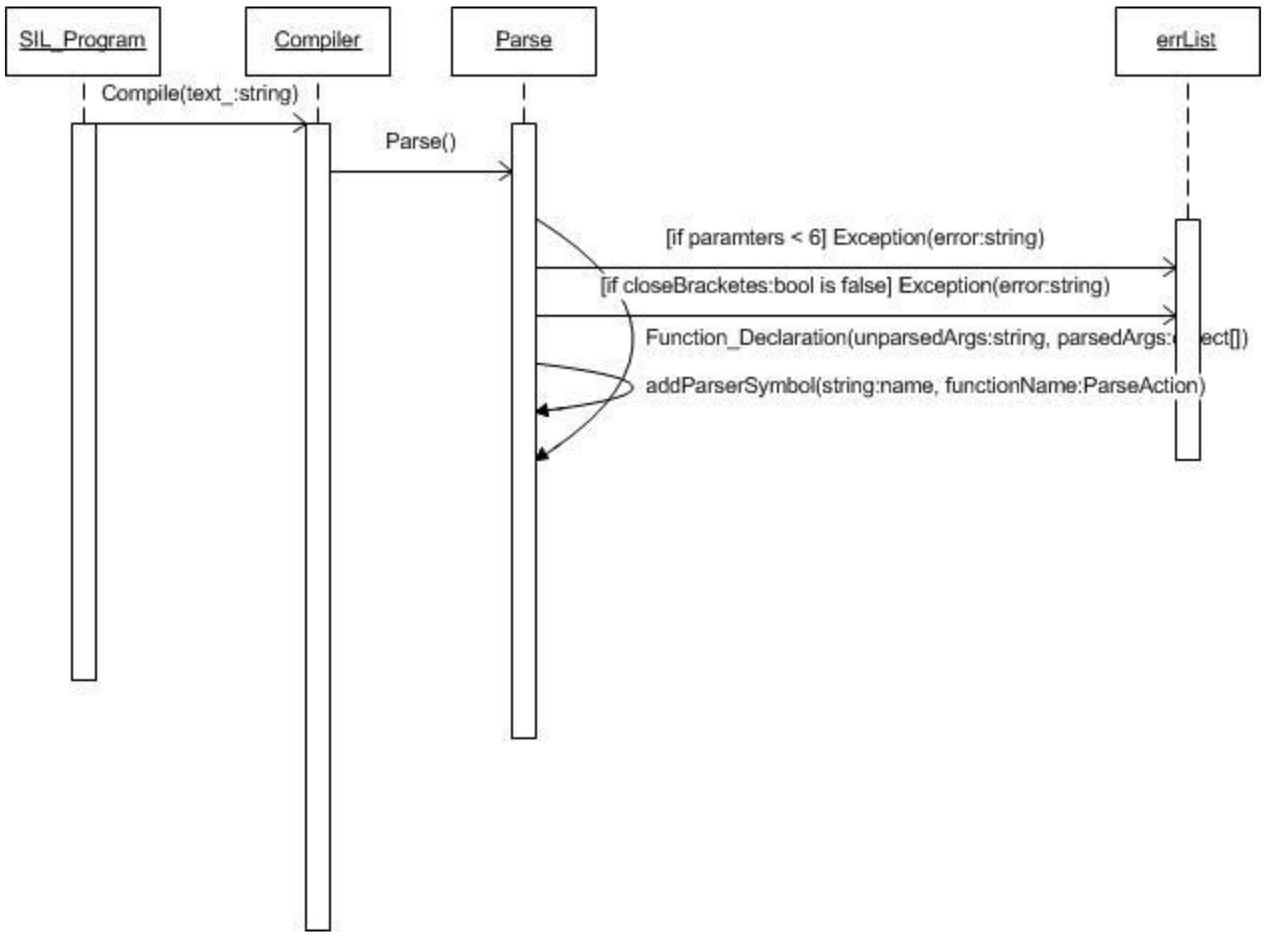
Expressions


SIL_Program
Compiler
SIL_Action
SymbolTable
Execute()
Execute()
EXEPRESSION_Execute(action:SIL_Action)
[aciotn.Args[0] = identifier]getValue(identifier:string)
return value:int
[if ac pty] EXEPRESSION_Execute(action:SIL_Action)
return value:int

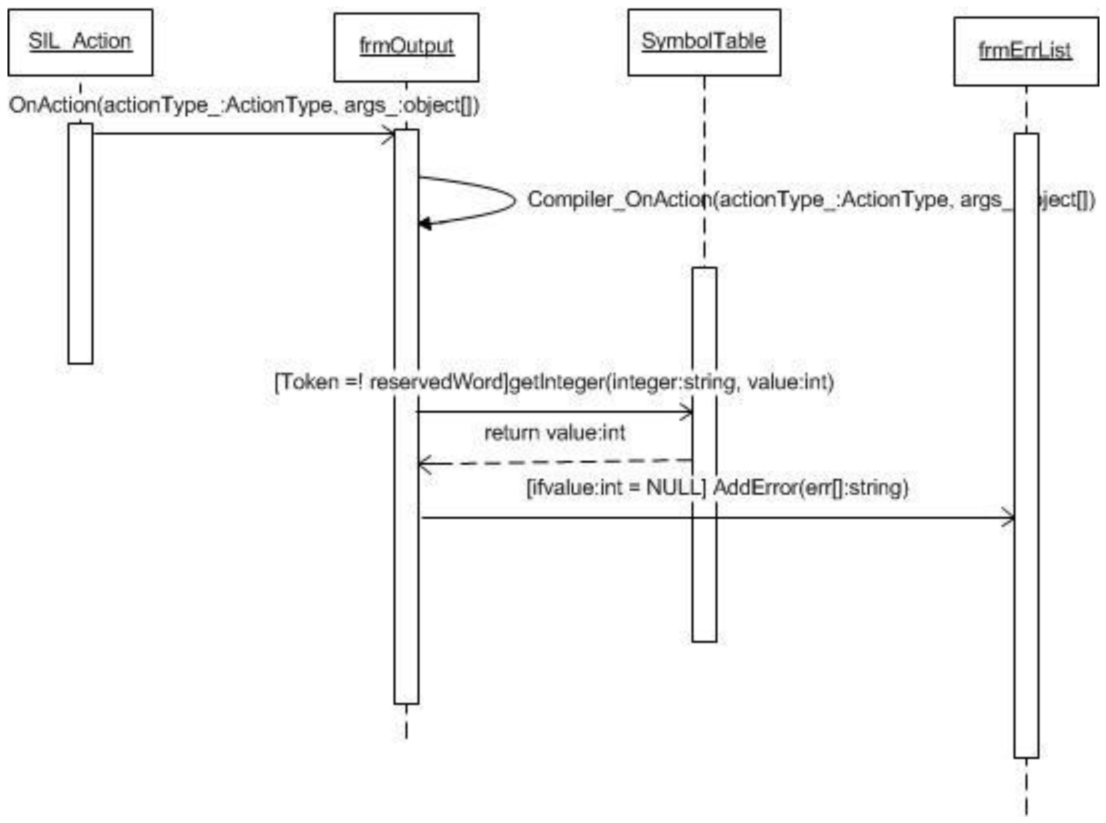
Function Call



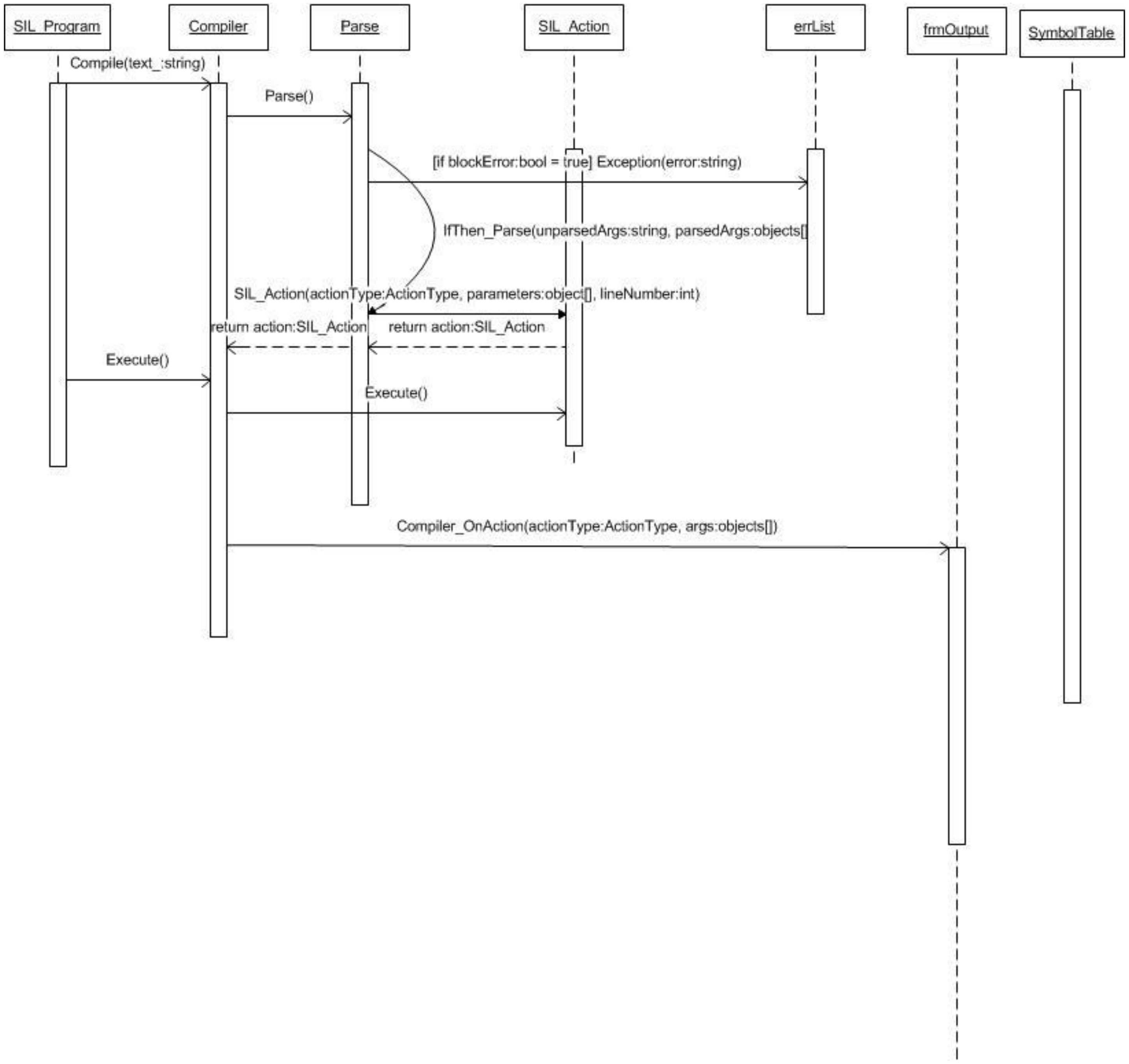
Function Declaration


SIL_Program
Compiler
Parse
errList
Compile(text_:string)
Parse()
[if paramters < 6] Exception(error:string)
[if closeBracketes:bool is false] Exception(error:string)
Function_Declaration(unparsedArgs:string, parsedArgs:object[])
addParserSymbol(string:name, functionName:ParseAction)

Identifier

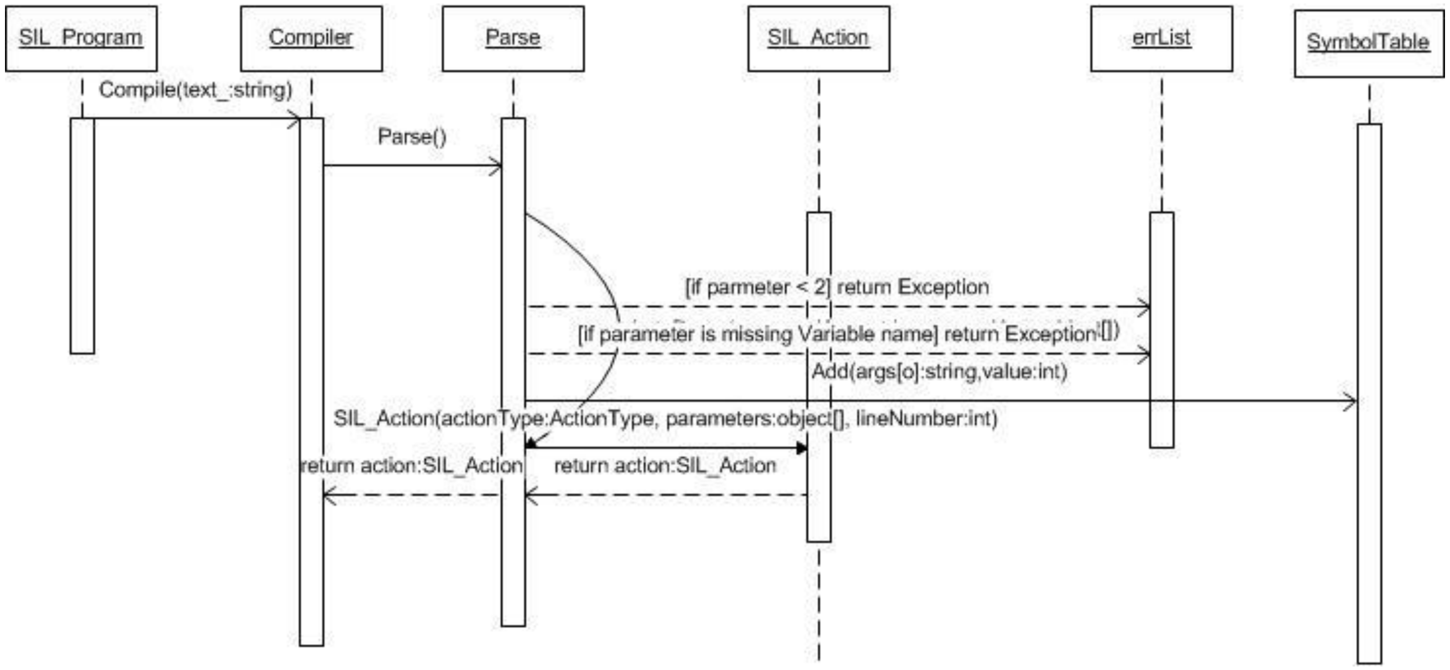

SIL_Action
frmOutput
SymbolTable
frmErrList
OnAction(actionType_:ActionType, args_:object[])
Compiler_OnAction(actionType_:ActionType, args_:object[])
[Token =! reservedWord]getInteger(integer:string, value:int)
return value:int
[ifvalue:int = NULL] AddError(err[]:string)

IF then Statement

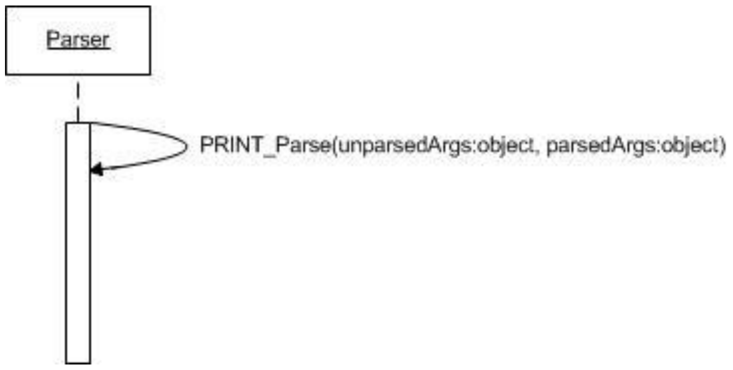


SIL_Program
Compiler
Parse
SIL_Action
errList
frmOutput
SymbolTable
Compile(text_:string)
Parse()
[if blockError:bool = true] Exception(error:string)
IfThen_Parse(unparsedArgs:string, parsedArgs:objects[]
SIL_Action(actionType:ActionType, parameters:object[], lineNumber:int)
return action:SIL_Action
return action:SIL_Action
Execute()
Execute()
Compiler_OnAction(actionType:ActionType, args:objects[])

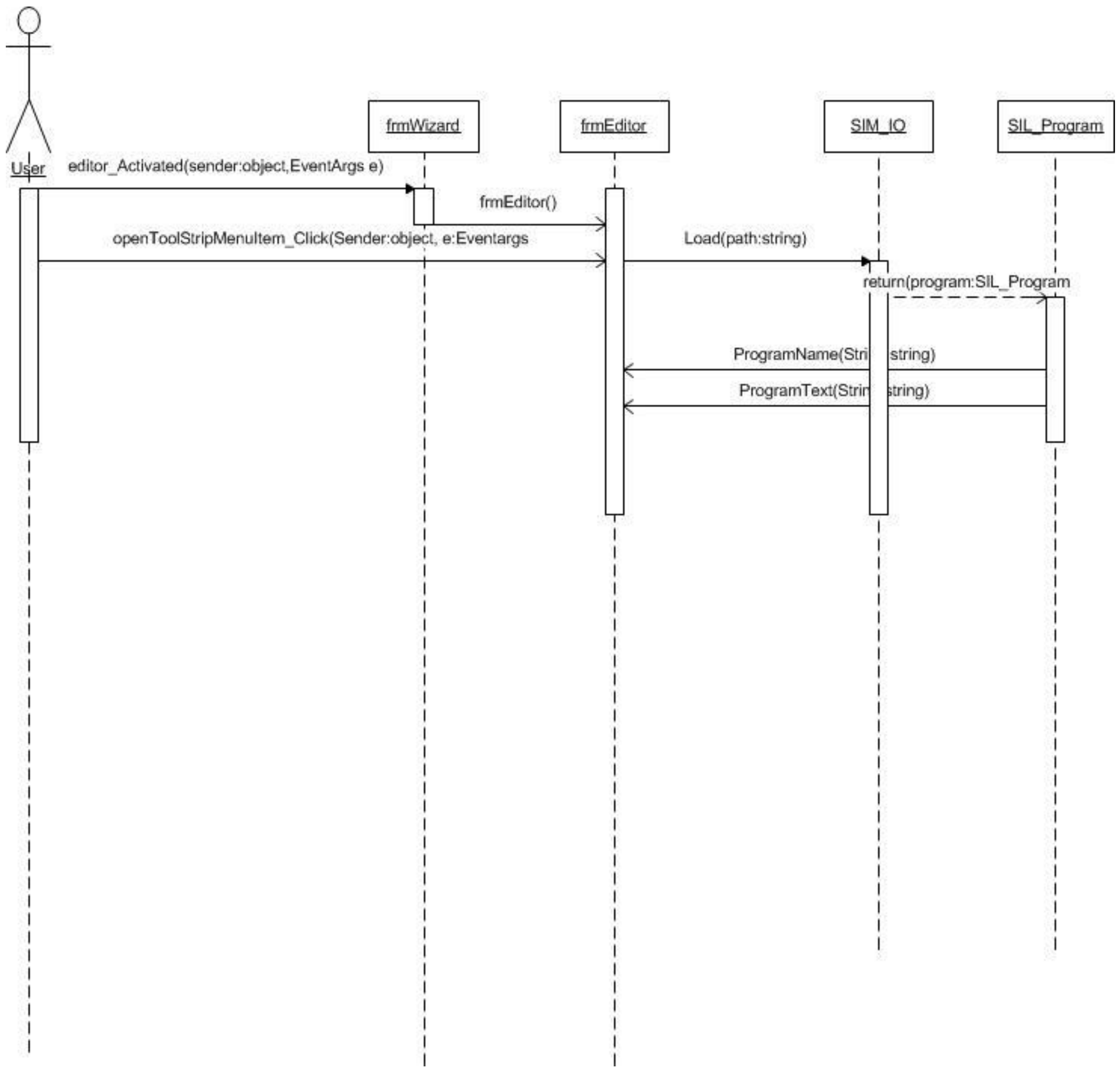
Let Statement


SIL_Program
Compiler
Parse
SIL_Action
errList
SymbolTable
Compile(text_:string)
Parse()
[if parmeter < 2] return Exception
[if parameter is missing Variable name] return Exception[])
Add(args[o]:string,value:int)
SIL_Action(actionType:ActionType, parameters:object[], lineNumber:int)
return action:SIL_Action
return action:SIL_Action

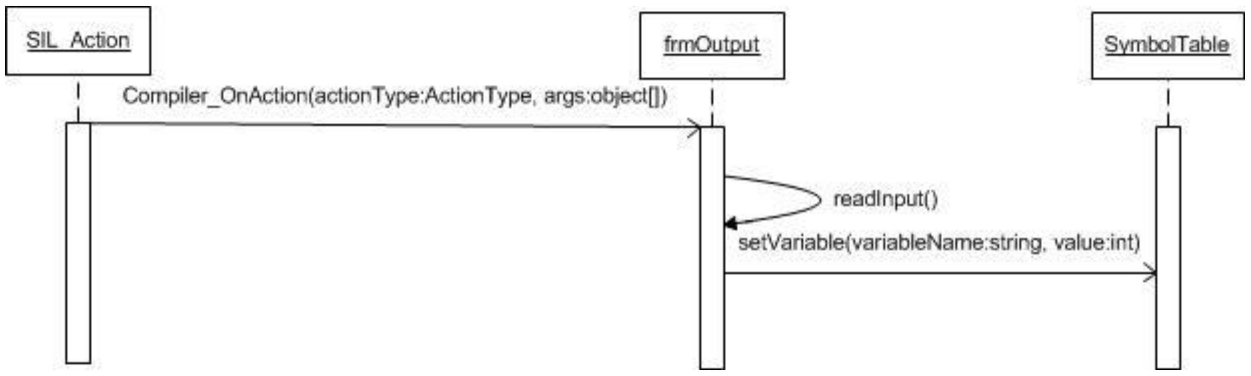
Literal


Parser
PRINT_Parse(unparsedArgs:object, parsedArgs:object)

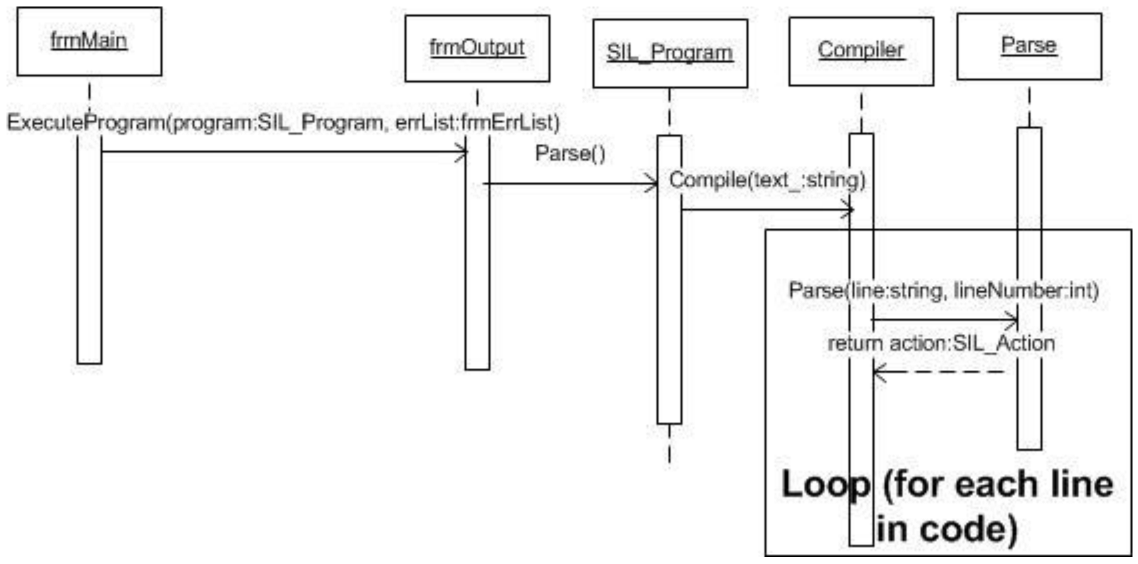
Load file into memory


frmWizard
frmEditor
SIM_IO
SIL_Program
User
editor_Activated(sender:object,EventArgs e)
frmEditor()
openToolStripMenuItem_Click(Sender:object, e:Eventargs
Load(path:string)
return(program:SIL_Program
ProgramName(Stri string)
ProgramText(Strir string)

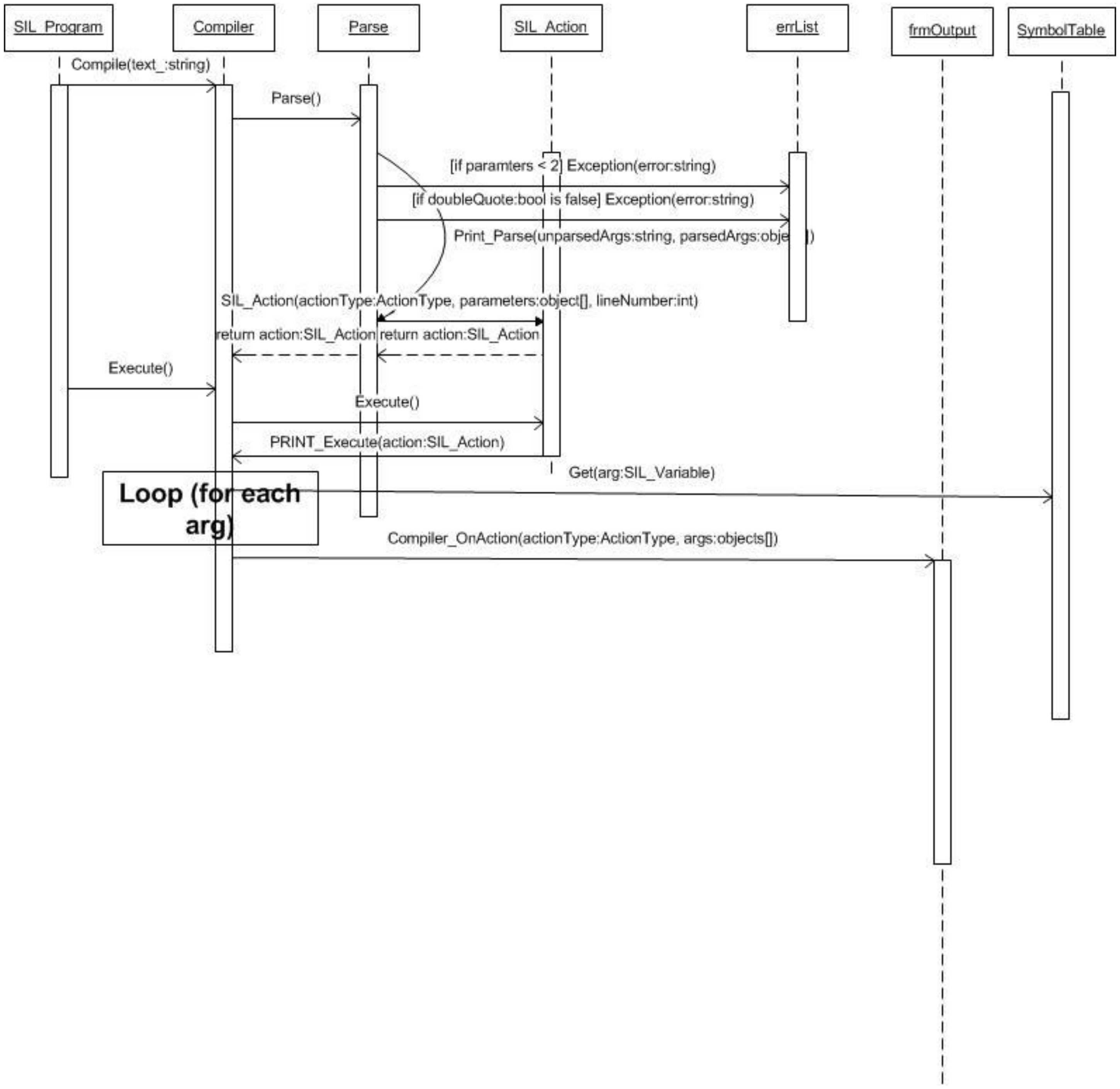
Obtain Input


SIL_Action
frmOutput
SymbolTable
Compiler_OnAction(actionType:ActionType, args:object[])
readInput()
setVariable(variableName:string, value:int)

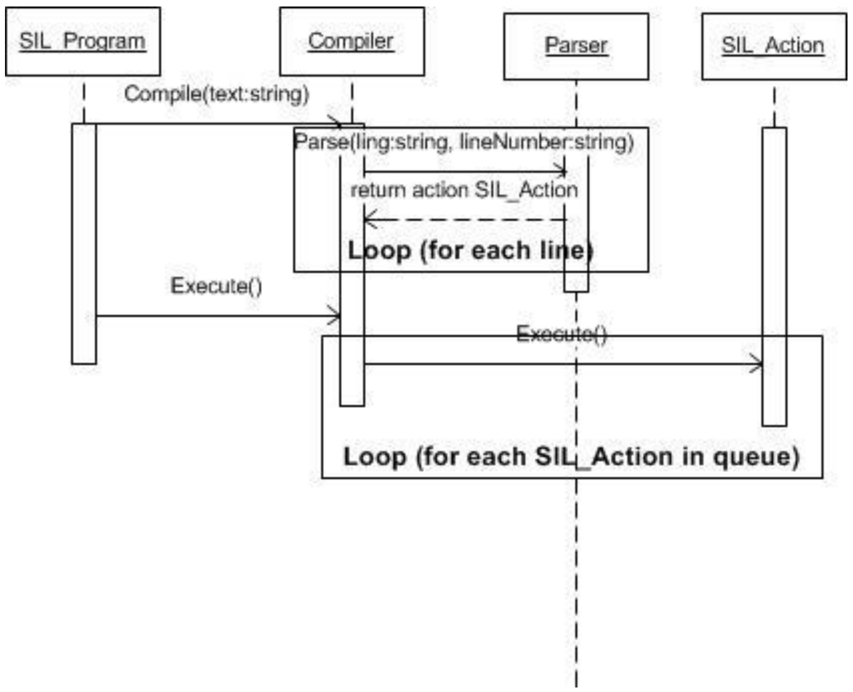
Parse the Program


frmMain
frmOutput
SIL_Program
Compiler
Parse
ExecuteProgram(program:SIL_Program, errList:frmErrList)
Parse()
Compile(text_:string)
Parse(line:string, lineNumber:int)
return action:SIL_Action
Loop (for each line in code)

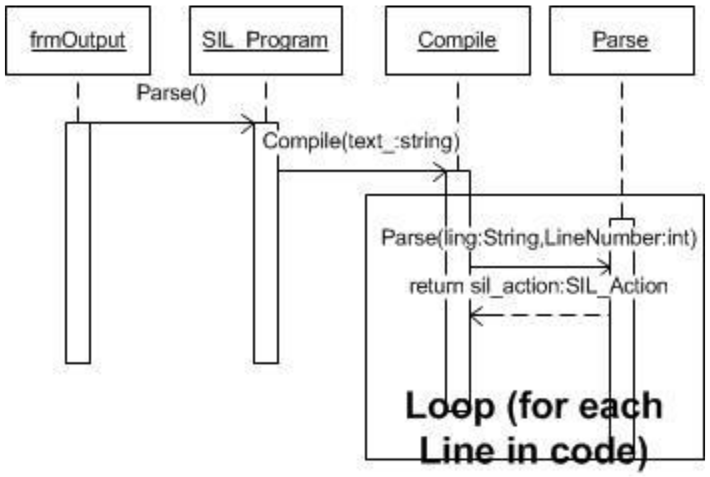
Print Statement


SIL_Program
Compiler
Parse
SIL_Action
errList
frmOutput
SymbolTable
Compile(text_:string)
Parse()
[if paramters < 2] Exception(error:string)
[if doubleQuote:bool is false] Exception(error:string)
Print_Parse(unparsedArgs:string, parsedArgs:obje   ])
SIL_Action(actionType:ActionType, parameters:object[], lineNumber:int)
return action:SIL_Action
return action:SIL_Action
Execute()
Execute()
PRINT_Execute(action:SIL_Action)
Get(arg:SIL_Variable)
Loop (for each arg)
Compiler_OnAction(actionType:ActionType, args:objects[])

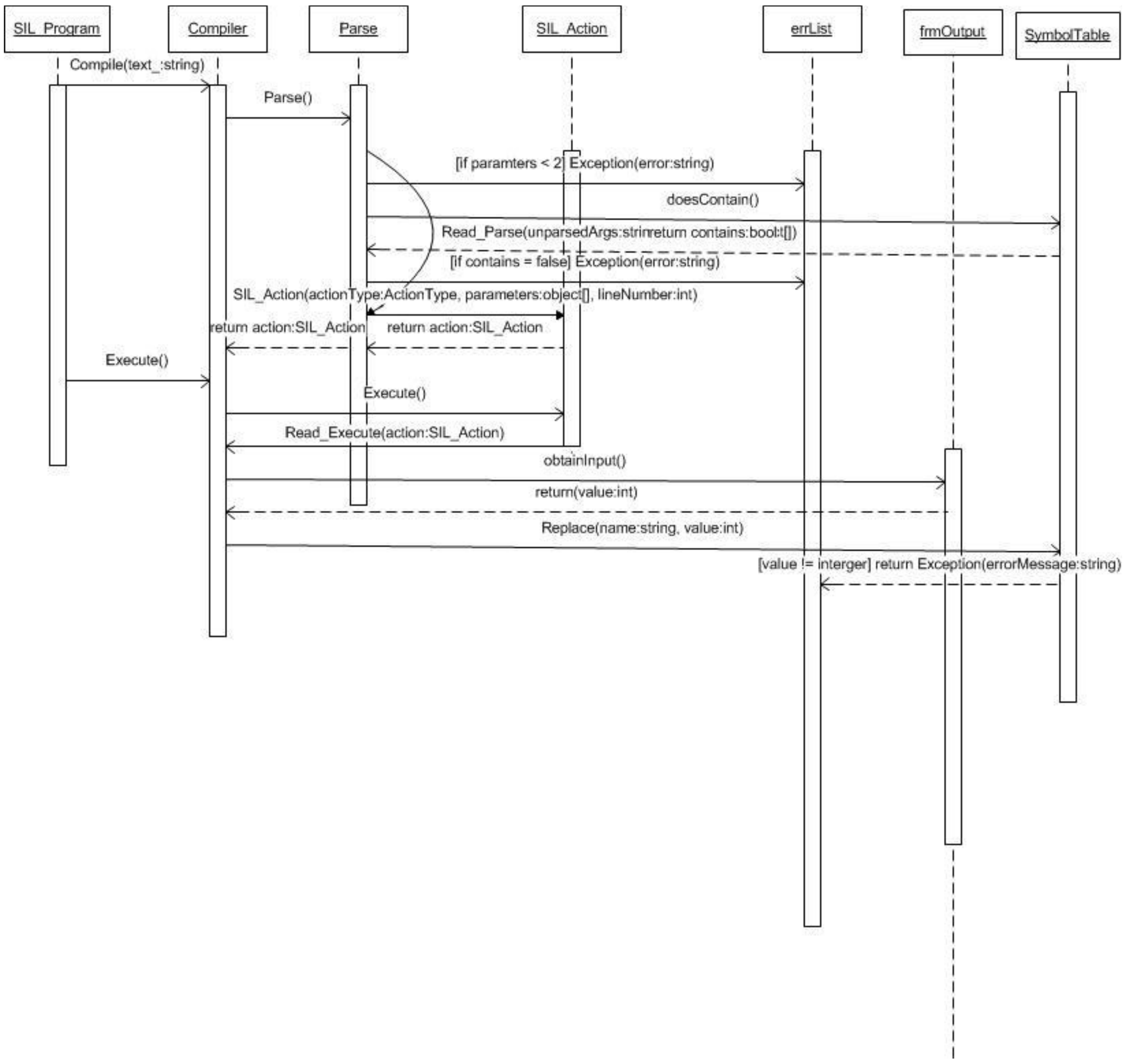
Read a Token Reserved Word



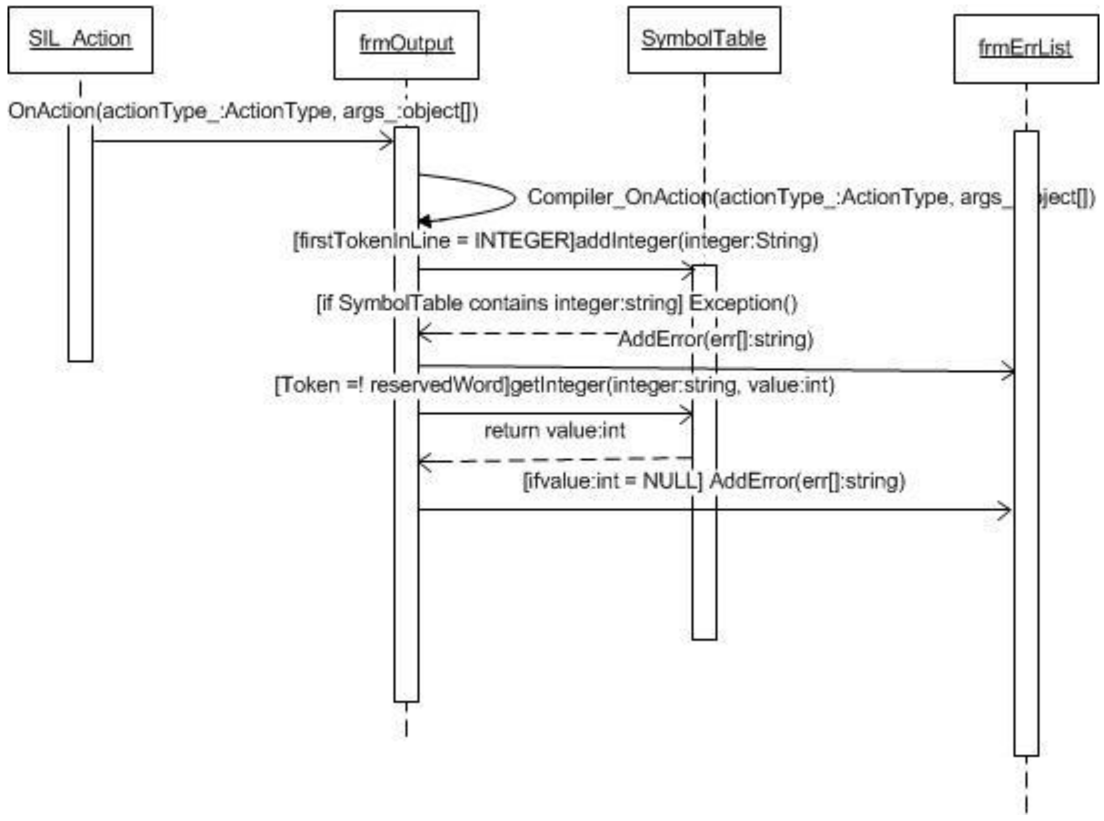
Read a token


frmOutput
SIL_Program
Compile
Parse
Parse()
Compile(text_:string)
Parse(ling:String,LineNumber:int)
return sil_action:SIL_Action
Loop (for each Line in code)

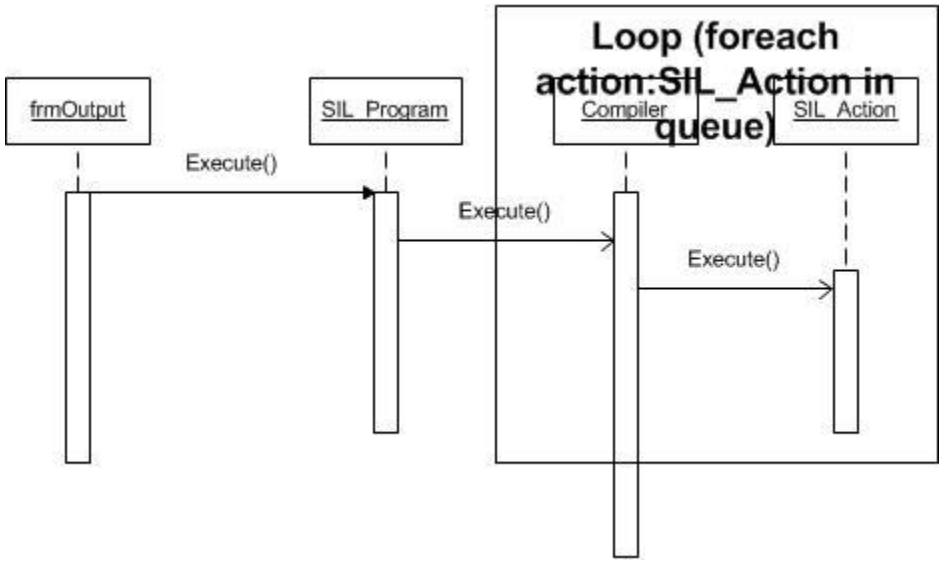
Read Statement


SIL_Program
Compiler
Parse
SIL_Action
errList
frmOutput
SymbolTable
Compile(text_:string)
Parse()
[if paramters < 2] Exception(error:string)
doesContain()
Read_Parse(unparsedArgs:strinreturn contains:bool:t[])
[if contains = false] Exception(error:string)
SIL_Action(actionType:ActionType, parameters:object[], lineNumber:int)
return action:SIL_Action
return action:SIL_Action
Execute()
Execute()
Read_Execute(action:SIL_Action)
obtainInput()
return(value:int)
Replace(name:string, value:int)
[value != interger] return Exception(errorMessage:string)

Symbol Table


SIL_Action
frmOutput
SymbolTable
frmErrList
OnAction(actionType_:ActionType, args_:object[])
Compiler_OnAction(actionType_:ActionType, args_ ject[])
[firstTokenInLine = INTEGER]addInteger(integer:String)
[if SymbolTable contains integer:string] Exception()
AddError(err[]:string)
[Token =! reservedWord]getInteger(integer:string, value:int)
return value:int
[ifvalue:int = NULL] AddError(err[]:string)

Translate Statement


Loop (foreach action:SIL_Action in queue)
frmOutput
SIL_Program
Compiler
SIL_Action
Execute()
Execute()
Execute()