DEVice **I**nterface **L**anguage (**DEVIL**)

Boklyn Wong (bw2007@cs.columbia.edu)
Pranay Wilson Tigga/Team Leader (pt2116@cs.columbia.edu)
Vishal Kumar Singh (vs2140@columbia.edu)
Hye Seon Yi (hsy2105@cs.columbia.edu)

“**A language that doesn't affect the way you think about programming is not worth knowing.”…..** *Anonymous*

Chapter 2: Lexical Conventions

2.1 Comments

The character # introduces a comment. Everything on that line will be considered as a comment

2.2 Identifiers

Identifier is a sequence of letter and digit with first character must be letter.

2.3 Keywords

The following identifiers are reserved for the language use and may not be used otherwise:

Object
Break
Continue
until
else
volatile
static
auto
if
options
while
default
tag
entry
repeat
int
if
char
repeat
extern
options
register
extern
done
return
template
done

2.4 Separators

( ) [ ] { } ;

Separators which are Ignored

Newline	\n
Tab	\t
Carriage	\r
White Space	

2.5 Types and Variables

Strong type controlled language. Mismatched types will not be automatically resolved or allowed by the compiler.

1. Data Types
   - Integer: It is a sequence of digits.
   - Character: 1 character.
   - String: It is a sequence of characters separated by “ ”. However, It is also equivalent to array of characters.
   - Boolean: The Boolean type has two values, represented by the strings “true” and “false”.
   - Array: The Array contains any of the supported data types.
   - Object: This data type comprises of other data types and is used to abstractly represent real world entities like devices.
2. Variables

   Each variable is of type of one of the supported data type.
   A variable has a name (Identifier) and a Scope.
   The scope can be
      I. Global.
      II. Local.
3. Initialization

   Integer to 0, character to ‘ ‘, String to NULL, Array to NULL.

4. Conversions

No conversion takes place from one type to another. Since the domain is Device configuration the control is in programmers/network administrators’ hand.

e.g.
Object FirewallDevice
a=FirewallDevice,
b = FirewallDevice.

E.g. int a;
e.g. string IPAddress;
IPAddress = a.b.c.d;

2.6 Operators, Declarations, Expressions, Statements and Blocks

2.6.1 Operators

The following is the list of Operators in our language.

```
< > <=  >=  ==   NOT
+   -  *    /    %
++  --  AND  OR
:=
@
->
>>
```

1. Operator + : Addition
2. Operator - : Subtraction
3. Operator * : Multiplication
4. Operator / : Division
5. Operator % : remainder of division
6. AND : Logical AND operation
7. OR : Logical OR
8. NOT : Not operation
9. ++ : Increment
10. -- : Decrement
11. = : Assignment
12. < : Less then
13. > : Greater then
14. <= : Less then equal to.
15. >= : Greater then equal to.
16. == : equal to

➔ This is for inheritance kind of relationship among object.
>> This is for composition kind of relationship among objects.

The **assignment** operator can be used for object assignments which would result in creating a new object in memory and assignments of all values except the values which are volatile.

The **NOT** operation is applicable only to Boolean type.

Multiplication, Subtraction, Addition, Division, and Remainder is applicable only to integer types.

The increment and decrement work on integer types only.

Unary Operator**:** Logical negation: NOT
NOT *expression;*

Postfix increment: (x ++)
Prefix increment: (++x)
Postfix decrement: (x--)
Prefix decrement: (--x)

Relational Operators

>	Greater then
<	Lesser then
>=	greater then equal to
<=	lesser then equal to

*expression * expression*
expression + expression
expression - expression
expression / expression
expression % expression

expression++
expression--
++expression
--expression

expression < expression
expression > expression
expression <= expression
expression >= expression
expression == expression

NOT e*xpression*
Identifier =expression
-> is for inheritance of object
>> is for composition of object.

E.g.

```
Object Router;
Object Firewall;
Object Port;
Object LinuxRouter;
Port p;
Firewall a;
Router r1;
LinuxRouter r2;
r2  -> r1 ;
```

```
# means router r2 inherits non –volatile characteristics of r1.
r2  >> a;
# by doing this we make firewall a part of router r2 so configuration on this
#device will automatically configure firewall too.
```

Operator Precedence:

NOT
OR , AND
*, /, % ,
++ , --, + , -
< , > , <= , >= , ==
=

2.6.2 Declarations

1. Object Type Declaration

    Declaration:

    Object Type_obj;

    Creates a copy of the type of object in memory. The object type must either be defined or should be available in library (if supported).

2. Object Instance Creation

    Declaration:

    Type_obj instancename;

    Creates an instance by copying from original object which is used for type definition.

3. Array Creation

    Declaration

    Type Identifier[expr];

4. Array Indexing

    Read
    Identifier[expr];
    Store
    Identifier[expr] = expr;

    # Assignment

    Indexing is used for storing and reading from Array position. If accessed position is null and error can be raised in runtime.

2.6.3 Statements

1. Expression Statements

   Most statements are expressions.
   Statements are executed in sequence.
   Successful evaluation of expression completes the statements.

2. Conditional Statement

   There are 2 conditional statements

   1. *if (condition) { statement }*
   2. *if (condition) {statement1 } else {statement2}*

   The *condition* in above 2 statements is an evaluation of *expression.*
   The statement in 1 is executed if condition is "true".
   The statement1 in 2 is executed if condition is "true" and statement2 is executed if condition is "false".

   The *else* is connected to innermost *if.*

3. Loop Statement

   - while statement
     while (condition) statement
     Condition is evaluation of an *expression.* When the expression becomes "false" the loop exits.

   - Repeat Until Statement
     repeat statement until(condition);
     The statement is executed until condition becomes false.

     The difference between *repeat-until* and *while* statement is that in repeat-until the condition check occurs after the execution of statement whereas in while it happens before execution of statement.

4. Break Statement

   The statement
   break ;
   causes termination of the smallest enclosing loop statement.

5. Return statement

   A function returns to the invoker by means of the *return* statement, which has following forms

   return ;
   return (expression) ;

In the first case no value is returned. In the second case, the value of the expression is returned to the caller of the function.

2.7 Functions

The form of function definition is described below.

function_definition:
return_type function_declaration function_body

function_declaration:
return_type function_name (parameter_list)

function_body:
local_declarations function_statements

function_statements:
statement_list

local_declarations:
var_type identifier

parameter_list:
identifier, parameter_list

statement_list:
statement statement_list

statement:
expression

return_type, var_type:
int,
char,
Object,
Array,
Boolean,
String

return_type is the data type which will be returned by the function.
var_type is the local variable type for the function.
local_declarations is the list of local variables.
function_statements consist of statement list which consist of expressions.
Every function has a return statement which returns one of the supported data types or void.

An example of function is

Declaration

```
Boolean  do_setHostname(string hostname)
```

Definition

```
do_setHostname(string hostname)
{
        string temp_hostname;

        generate(“sethostname”,hostname);

        return  true;
}
```

In our language functions are like logical components of a big task which are mapped in order to achieve the completion of a task.

All the functions are global.

2.8 Future work: Ontology

The team is working to devise a way to represent the knowledge which a device has and other devices can reuse using language constructs which are intuitive to network admins. This may include representing things like relationship between different kind of devices and there interactions. E.g. a Linux based router can inherit knowledge from a generic router. But the same router can have ports and can also act as firewall device. The relationship between Linux Router and Firewall is of Association, relationship between port and Linux router is composition and relationship between Linux Router and Generic Router is of Inheritance. Establishing these relationships will enable information sharing between devices.

2.8.1 Sample Code

Prg1.dv

```
# This program sets the hostname of a linux machine

Object LinuxMachine {

static string  platform = “Linux”;
volatile string hostname;

}
```

```
boolean set_hostname(string hostname)
{
            string command =  “sethostname”;
            genshell( command , hostname);
            return true;
}

Main( )
{

            LinuxMachine lm ;
            set_hostname (“PLT_LAB”);

}
```

O/p is a file with following entry :
sethostname PLT_LAB

Prg2.dv : Enable firewall on Linux

```
Object LinuxMachine {

static string  platform = “Linux”;
volatile string hostname;

}
Object LinuxFirewall;
LinuxFirewall -> LinuxMachine;

LinuxFirewall {
     boolean enabled = false;
}



boolean set_hostname(string hostname)
{
            string command =  “sethostname”;
            genshell( command , hostname);
            return true;
}
boolean enable_firewall( )
{
     string command = “ipchain enable firewall”;
     genshell(command);
}

Main( )
{

        LinuxFirewall lm;
        set_hostname (“PLT_LAB”);
         enable_firewall ( );
}
```

O/P

```
Sethostname PLT_LAB
Ipchain enable firewall
```