iCalendar Language Reference Manual

1. Lexical Conventions

In iCalendar there are several classes of tokens could be supported. Token types are identifiers, keywords, literals, strings and operators. As in C language, whitespace characters are ignored except insofar as they serve to delineate other tokens in the input stream. If the input stream has been parsed into tokens up to a given character, the next token is taken to include the longest string of characters which could possibly constitute a token.

1.1 Notation

Through the document, *nonterminals* are in italics and **terminals** are bold format. Regular expression-like constructs are used to simplify grammar presentation.

- r* means the pattern r may appear zero or more time. r+ means the r may appear one or more times.
- r? means r may appear zero or once.
- r1 | r2 denotes an option between two patterns.
- r1 r2 denotes an option between two patterns, r1 r2 denotes r1 followed by r2.

1.2 Comments

In iCalendar, we use # to mark comments which means a line of comments should be started with the character # and ends with another character #. However, in iCalendar only one-line comments could be supported, that means if the length of comments exceeds one line, remember to put # at the beginning of each line.

Accept:
#This is a comment#
#This is another#
#line of comment#
Not accept:
#This is another
line of comment#

1.3 Identifiers

An identifier consists of a letter or an underscore followed by other letters, digits and underscores. Identifiers are case sensitive, so “left” and “Left” are different identifiers. The length of an identifier is not limited.

Accept: _; _a; a11; b; b12; C23fjdla;
Not accept: 123; 12abc;

In iCalendar global and local variables are both supported. A variable defined inside a block is local, and cannot be called outside the block. For example:

```
int Add(int a, int b)
  {
 Int c = 0;
 return a+b+c;
  }
```

```
int b = c; #This is illegal
```

1.4 Keywords

In iCalendar, the words listed below are reserved as keywords. Users are not allowed to define their own identifiers.

Reserved identifiers in iCalender:

int	float	string	bool
true	false	if	else
while	for	Event	Calendar
return	void	main	
size	print		

1.5 Constants

In iCalender, several kinds of constants including integer constants, float constants, string constants and boolean constants are supported.

Integer constants

Integer constants should be decimal and consists of a sequence of numbers without a decimal point. All integers should be unsigned.

Accept: 45, 0

Not accept: -1, +12, 1.5

Float constants

A floating-point constant consists of a sequence of numbers and a decimal point. Before the decimal point is the integer part and the decimal part is after the point. The integer part could be omit. Examples of valid and invalid cases are listed:

Accept: 0.5; .234; 10.55

Not accept: 1 1.; .; 1.1.1; 2..5

String constants

String constants are demarcated by double quote characters, for instance “iCalendar”.

Escape characters are supported.

Character Name	Escape Sequence
Newline	\n
Horizontal tab	\t
Backslash	\\

Boolean constants

Boolean constants consist of the keywords true and false. For example: bool b = true

2 Types

In iCalendar, seven fundamental types of objects are supported, but no type conversion is allowed. The fundamental type of objects are a following:

- int
- float
- string
- bool
- Event
- User defined Event structure
- Calendar

Integer type

In iCalendar the only supported integer type is int which can store 32-bits worth of data. This data type is signed.

Float type

In iCalendar the only supported double type is double which can store 64-bits worth of data. Tis data type is signed.

String type

In iCalendar we provide string type, which means a string of unlimited length could be supported. However, the length may be limited because of the amount of computer resources.

Bool type

In iCalendar bool type is supported. Bool type could only take a value of either true or false. However, bool type is very useful and helpful.

Event type

Event type is to enable users to define their own event structure.

User Defined Event Structure

User could define their own event model to use for recording events. For example, Use use Event type to define:

```
Event myOwnEvent
        { int time;
        string description;
        int priority;}“.
```

Calendar type

Calendar is like a list to contain events.

3 Expressions

In iCalendar, expressions consist of operators and their operands. In this section, definition for each operator is given. To avoid ambiguity, precedence rules of the operators in iCalendar are also defined in this section.

3.1 Operators and Punctuations

Operator	Description
+ - * /	plus/minus/multiple/divide
> = != < >= <=	greater than/equal/not equal/less than/greater than/less than
&& || !	logical and/logical or/logical not
.	dot
,	comma(separate expressions)
;	end of expression
() []	separators

Tokens	Operators	Class	Associativity
* /	Multiplicative	Binary	L
+ -	Additive	Binary	L
< <= => >	Relational comparisons	Binary	L
== !=	Equality comparisons	Binary	L
&&	Logical AND	Binary	L
||	Logical OR	Binary	L
!	Logical NOT	Unary	R
=	Assignment	Binary	R
,	Comma	Binary	L
.	Dot	Binary	L
;	end of expression	Unary	L
() []	separators	Unary	L

3.2 Primary Expressions

Identifiers, constants, strings. The type of the expressions depends on identifier, constant or string.

3.3 Arithmetic operators

In iCalendar, arithmetic operators are +, -, * and / . + means addition, - means subtraction, * means multiplication and / means division. All of them are binary and left associative. It requires that their operands must be of the same primitive types, and the result will be of the same type.

3.4 Comparative operators

In iCalendar, comparative operators are > (greater than), < (less than), >= (greater than or equal to), <= (less than or equal to), ! = (not equal) and == (equal). All of them are binary operators and left-associative. It requires that their operands must be of the same primitive types . The return value is a boolean value indicates the predicate.

3.5 Logical operators

Logical operators in iCalendar include && (logical and), || (logical or) and ! (logical not). && and || are binary operators and left-associative. They take two operands of type boolean, and return a boolean value. ! is unary and appears on the left side of the operand. They type of the operand must be of type boolean and the return type is also a boolean value.

3.6 Assignment operators

iCalendar's assignment operator is =. It's a binary operator and right-associative. The left operand must be a legal left value, and the right operand must be an expression. When an assignment is taken, the value of the expression on the right is assigned to the left value, and the new value of the left value is returned.

3.7 List operations

In iCalendar, users could store their event objects to a calendar.
A calendar is like a list data type. And to visit the element in calendar, use index (e.g. calendar[0]).

3.8 Dot operators

User defined event model might have many attributes (like time, event description, location, event priority). These values can be visited by using dot ".". For example, we have an object: myEven e = ["final exame","Dec-10"]. The time can be visited by e.time where "time" is an attribute of myEven type defined by users.

4 Statement

Statements in iCalendar contain expressions, return statement, return void statement, conditional statement, loop statement, variable declarations, empty statement, etc.

Statement ->
Block*
| Expr*
| Return*
| ReturnVoid*
| If statement*
| For statement*
| While statement*
| Vardecl*
| Empty*

4.1 Expressions

An expression statement is composed of primary statements with a semicolon at the end of the line. It is used for binary operations.

4.2 Return Statements

A compute function returns a value to the caller through return statements.

Return Statement -> **return** Expression **SEMICOLON**

4.3 Returnvoid Statements

This function returns a void value to the caller through return statements.

Returnvoid Statement -> **return** void **SEMICOLON**

4.4 If Statements

iCalendar supports two kinds of if-else statemens:

If Statement ->
If (Boolean conditions) Statement
|**If** (Boolean condition) Statement **else** Statement

4.5 For Statements

In iCalendar, users could use For statement.

For Statement -> **for** (Expr **SEMICOLON** Expr **SEMICOLON** Expr) Statement **SEMICOLON**

4.6 While Statements

C programmers are often tolarate with for statement, as they have to type in verbose statement. While, in iCalendar, to avoid verbose, we just use while as the loop statement.
While Statement -> **while** (condition) Statement

4.7 Variable Declaration Statements

In iCalendar, if a line contains just a semicolon, that means a null statement and has no meaning.

Variable Declaration -> datatype (**Identifier** (Assign Expr)? **COMMA**)*Identifier (Assign Expr)? **SEMICOLON**

4.8 Empty Statements

In iCalendar, if a line contains just a semicolon, that means a null statement and has no meaning.

5 Grammar

5.1 Program Definition

A program in the iCalendar language consists of a sequence of Event definition structures, variable declarations and functions executing in order.

Program -> Even_Definition_list * Declaration_list * Function_list*

5.2 Event Definition

Users could define their own event that consists of a sequence of type name and variable declarations executing in order.

Event -> Typename_string*Declaration_list*

5.3 Declarations

All variables must be declared before they can be used. However, variable declarations can be made at any point in a program. Variables become usable after the end of the semi-colon of the statement in which it contained.
In iCalendar, declarations consist of variable declarations and function declarations.

Declaration -> Variable Declaration*
|Function Declaration

Calendar Language supports functions. Function Declaration is definte below:

Function Declaration -> returnType **Identifier** ((datatype variable)*
|(datatype variable **COMMA**)+ datatype variable) Block

For example:

```
fun int Add (int a, int b)
{
    return a+b;
}
```