TScript Manual

A simple Script Language for database

Author & maker:

Niklaus F. Schen

Contents:

Preface

TScript is a simple and purely procedural script language. It can be learned more easily if user is a C programmer. Now, This language is only used on a private database.

As a kind of programming language, the variable is an important concept. It will be discussed in Chapter 1.

TScript supports several data types. Those are compatible with C language. We will discuss them in Chapter2.

Next is the traditional concept -- *Function*. Actually, there is no function definition and declaration. There are only six built-in functions -- *fopen*, *fclose, query, print, lseek* and *eof* just like *sizeof* in C language. We will describe it in detail in Chapter 3.

In Chapter 4, we will talk about operators and their precedence. These operators are a subset of C's. In addition, type *string* can be supported to execute addition ,subtraction and part of logical operations in TScript.

In Chapter 5, *expression* will be discussed. We will focus on its usage scenarios. And the last Chapter, we come to talk about *statement*.

There are some examples in Appendix. User can refer to them to write your own script.

Variable

Let us begin with an example:

(a)int i = 99;

This *i* we call it *variable*. How can we recognize a variable? That's quite easy. Firstly, variable's name should be consisted of letters, digits and underlines, and the first one should be a letter or underline.

Variable definition is consisted of two parts at least: type and name, like form (b). And sometimes it will be initialized by a specific value, we call this initialization *assignment*, just like form (a). More important, variable must be defined before we use it (showed in (b) and (c)).

(b). int j; //definition

(c). j = 1; //assignment (a kind of usage)

The *int* is a kind of *type*, we will discuss it in Chapter 2. It points out the variable which followed *int* (in this case is *j*) is an integer. And the character =, that is a kind of operator, which is used to assign a right-value (in this case is *99*) to the left variable. Actually, the right-value not always be a constant, it also can be a variable or something else. For now, TScript only supports decimalism and hexadecimal value.

As a variable, it may be changed many times. So we need a space to store it. This space, in fact, is allocated in the environment list that we don't talk about in this manual.

Besides storage space, there is another concept -- *domain*. Every variable has its lifetime that depends on the position of its definition. If we define every brace closure as a new domain, we can define the lifetime for every variable. There is an example:

if(i<100) { int k; };

We do not talk about *if* until the Chapter6. People just need to know that if variable *i* is satisfied with the condition(in this case is *i<100*), program will jump into the brace closure. Thus, there is an integer named *k* will be created. So what is the *k*'s lifetime or the domain it belongs? When *k* is created, its lifetime will begin at the same time. And when the procedure jump out the right brace, *k*'s domain will be gone, and we can not track this variable any more.

Data Types

Data types usually indicate the size of numbers or variables, but there are some exceptions. TScript prohibits type definition, so there is only seven data types in this script language. See table 1.

TYPE	SIZE	RANGE
char	1byte	-128~127
short	2bytes	-32768~32767
int	4bytes	-2147483648~2147483647
long	8bytes	-9223372036854775808~9223372036854775807
float	4bytes	-2^127~-2^-129, 2^-129~(1-2^-23)*2^127
string	No limit	N/A
file	No limit	N/A

Table1. All data types

Let us describe them sequentially.

char -- defines a smallest integer value of all data types. There is only one byte it takes. Its range is -128 to 127 in decimal. But this type is usually used to print characters. People can find them in ASCII table. Here some definition forms are given:

char c; or *char c = 10;* or *char c = 'a';*

short -- defines a longer integer value than *char*, but it still be smaller than a regular integer size. It has 2 bytes, and its range is -32768 to 32767. Variable definition forms:

short sh; or *short sh = 2;*

int -- defines a regular integer value. This value has 4 bytes, and its range is -2147483648 to 2147483647 in decimal. We can such define a variable:

int i; or *int i = 45000;*

long -- defines a longest integer value of all integer types. It has 8 bytes and its range is -9223372036854775808 to 9223372036854775807.

Variable definition forms:

long l; or *long l = 9999;*

float -- defines a float value. It's a little bit complicated. But we just need to know that this type is used to define a decimal, and it has two ranges: -2^127 to -2^-129 and 2^-129 to (1-2^-23)*2^127. Variable definition forms:

float f; or *float f = 10.33;*

string -- defines a string value whose size is not sure. We do not need to restrict a size for a string variables. TScript will allocate and free memory blocks by itself. So this feature allows arbitrary length of the string assigned to a string variable. Variable definition forms:

string s1; or *string s1 = "hello";*

file -- defines a file descriptor. It's very like type *FILE* in C language. When we define a file variable without calling function *fopen()*, the variable will equal to a special file variable *ferr*. So that means, we must open file successfully before we use it. Here some definition forms are given:

file fp; or *file fp = fopen("/tmp/test");*

Above all, TScript only supports seven data types. It's no quite difficult, especially for the C programmer.

Functions

Just like what we said in Preface, TScript do not support function definition and declaration. But we need some things to control and manage file, and we also need at least one mechanism to do some special things that we want. So there are six built-in functions provided. They are *fopen, fclose , query, print, lseek* and *eof*.

fopen -- This function is used to open a file. Here is the function prototype:

file fopen(string path);

The argument *path* is a string value, which represents a file path we want to open, and its return value is a new *file* variable. If the file is not existed or some other errors happened, the return value would be equal to *ferr*. We can call function *print* with argument *%e* to print error information in terminal. People can find some examples in Appendix. Program will close the opened files automatically when procedure is done.

fclose -- This function is used to close a file. Function prototype:

void fclose(file f);

The argument *f* is a *file* variable, but we do not require it valid. Which means, we can call this function even if the argument is equal to *ferr*, and the program won't get an error.

query -- This function is used to execute lots of operations for anything user wants (in this case is the database operations). Function prototype:

int query(string s, ...);

The return value is an integer that represents success (0) or failure (-1). About the arguments. The first one is a string value which records the commands that user want to execute. The second argument is *...,* which means there may be more than one or no arguments followed.

Here we need to talk about the format control, just like "%d%f%s" in C language. TScript supports five format control characters. See table 2.

%d	Represents an integer value.
%f	Represents a float value.
%c	Represents a character.
%s	Represents a string value.
%e	Represents an error message.

Table 2. Format control characters

Example:

query("%d %s %c %f\n", 120, "hello", 'd', 3.1415926);

print -- This function is used to print informations in terminal. Function prototype:

string print(string s, ...);

The function's arguments are same as *query*, so we don't discuss it any more. And the return value, I decide to make it be a *string* value. Thus, we can use this return value to query or assign to another *string* variable. This value is equal to the first argument which has been transformed by format control characters.

lseek -- This function is used to set the current offset in an opened file. Function prototype:

int lseek(file f, int offset, string whence);

The argument *whence* has two valid values: *SEEK_SET* and *SEEK_END*.

SEEK_SET: The offset is set to *offset* bytes.

SEEK_END: The offset is set to the size of the file plus *offset* bytes.

Examples:

lseek(fp, -3, "SEEK_END");

lseek(fp, 0, "SEEK_SET");

eof -- The return value of *eof()* is just the opposite with *feof()*'s return value in C. Function prototype:

int eof(file f);

The function eof() tests the end-of-file indicator for the stream pointed to by *f*, returning zero if it is set.

Operators

In TScript, there are 22 operators. Here is a table we given to show:

Priority	Operator	Meaning	Number of objects	Combined direction
1	( )	Parentheses	N/A	Left->right
2	!	NOT	1	Right->left
	~	Negation		
	-	Negative sign		
3	*	Multiplication	2	Left->right
	/	Division		
	%	Remainder		
4	+	Addition	2	Left->right
	-	Subtraction		
5	&	Bitwise AND	2	Left->right
	|	Bitwise OR		
	^	Bitwise XOR		
6	&&	Logical AND	2	Left->right
	||	Logical OR		
	==	Logical EQUALITY		
	!=	Logical INEQUALITY		
	>	Greater-than		
	<	Less-than		
	>=	Greater than or equal		
	<=	Less than or equal		
7	=	Assignment	2	Left->right
8	,	Comma	N/A	Left->right

Table 3. Operator table

We can see, there are 8 priorities in this table.

Number of objects -- means the number of objects which participate in an operation.

Expression

Expression is a valid sequence composed of variables, constants and operators in TScript. There are many places we use expression. Here we present some productions belong to TScript:

(a) tydecl-> ty factor assign;
assign-> EQ exp;

(b) select_exp-> IF LPAREN exp RPAREN LBRACE stm RBRACE elsee;

(c) loop_exp-> WHILE LPAREN exp RPAREN LBRACE stm RBRACE;

(d) func_decl-> QUERY LPAREN exp RPAREN;
| FOPEN LPAREN exp RPAREN;
| FCLOSE LPAREN exp RPAREN;
| PRINT LPAREN exp RPAREN;
| LSEEK LPAREN exp RPAREN;
| EOFF LPAREN exp RPAREN;

(e) explist-> ;
| COMMA exp;

(f) exp-> func_decl explist;
| assignexp explist;

The *exp* is the expression which composed of two productions, see (f).
One production is *func_decl* allowed the functions' return value to be an expression, see form (1). But the form (2) is illegal.

(1) *if(lseek(fp, 0, "SEEK_SET")) {...};*
(2) *if(lseek(fp, 0, "SEEK_SET")<0) {...};*

Which means, the function's return value only can be a full expression, not a part.

The other one is *assignexp* allowed user to write the sequence which we have talked at the beginning of this chapter.

The *explist* means expression list appeared at the end of *exp* productions. It allows a new expression to follow the previous one separated by a comma.

Production (a) is a assignment production, we can see the right-value can be an expression, not only a variable or constant. And production (d) also means the arguments can be an expression, not only a variable or constant.

Production (b) and (c) are similar. Their e*xp* allows the judgment condition to be an expression.

More examples will be given in Appendix.

Statement

What is the statement? How to define it?

Statement is a minimized unit which can be run correctly on TScript. Here are the productions of statement:

(a) start-> stm ENDFILE;

(b) stmlist-> ;

| SEMIC stm;

(c) stm-> ctlexp stmlist;

| tydecl stmlist;

| select_exp stmlist;

| loop_exp stmlist;

| func_decl stmlist;

| ;

(d) select_exp-> IF LPAREN exp RPAREN LBRACE stm RBRACE elsee;

(e) elsee-> ELSE LBRACE stm RBRACE;

| ;

(f) loop_exp-> WHILE LPAREN exp RPAREN LBRACE stm RBRACE;

The *stm* represents statement. So (c) is the statement definition in TScript. We can see, statement is composed of 6 productions. Each statement is separated by a semicolon (see (b)).

Now, here are six kinds of statement we will introduce:

(1) Control statement -- this kind of statement includes three keywords: *break, continue* and *return*. The first two statements are used in loop statement. *Break* is used to jump out from a loop procedure. And *continue* is used to go back to the beginning of a loop procedure. The last one *return* can make the procedure stop and return. We can write according to the following forms:

break;

continue;

return;

(2) Variable definition and assignment -- this kind of statement is quite easy, let us see some examples:

```
int i = 12;
string s = “abc”;
file f;   f = fopen(“/etc/abc”);
char c = ‘d’; c;
```

(3) Select statement -- IF-ELSE statement. Let us begin with two examples:

```
(a)  if( i<10 ) {
        i = i+2;
     };
(b) if( i<10 ) {
        i = i + 2;
     } else {
        i = i - 2;
     };
```

The first one only has *if*, and the second one has *if* and *else*. There is a difference between TScript and C language. That is the semicolon at end of the statement. C does not have this semicolon.

There is a condition in the parentheses. If it is true, the procedure will jump into the adjacent brace closure, or the procedure will jump into the *else* brace closure if the statement has an *else*.

(4) Loop statement -- there is only one loop statement *while*.

```
while( condition ) {
    statements; ...
};
```

If the *condition* is true, the procedure would keep looping. More examples are given in Appendix.

(5) Function statement -- Actually, this is not a kind of statement, because functions also belong to *expression*. But we need a way to call a function without using its return value. We have discussed functions in Chapter 3, so we don’t discuss it here any more.

(6) Empty statement -- This statement has only one semicolon. Just like:

```
;
```

It means nothing actually.

Appendix

Here are some examples, hope these are useful.

Example 1:

```
int score;
int age = 30, score = 97; /*this score is not a definition, but a assignment.*/
/*we do not allowed:
* int arg = 30, tmp = 99;
* to define variable tmp.
*/
char sex = 'm';
string name = "John";
float money = 23045.39;
print("information: name[%s] age[%d] sex[%c] money[%f]\n", \
                                                    name, age, sex, money);
```

Example 2:

```
int i = 0;
while( i<10 ) {
    if( i>5 ) {
        int i; i = 100;
        print("i in if is:%d\n", i);
    };
    print("i: %d\n", i);
    i = i + 1;
};
```

Example 3:

```
float f = 3.1;
if( f<9.9 ) {
    print("less than 9.9\n");
} else {
```

```
        print("greater than 9.9\n");
    };
```

Example 4:

```
    string str1 = "hello";
    string str2 = " world";
    string str3;
    str3 = str1 + str2;
    string str4 = print("%s\n", str3);
    print("again:[%s]\n", str4);
    print("str3 - str1 = %s\n", str3-str1);
    if( str3!=str4 ) {
        print("fatal error.\n");
        return;
    };
    print("Good!\n");
```

Example 5:

There is a file named *test*, it's content is:

John,"this is a csv file, and I only recognize this format.",20140108

Here is the content in script file:

```
    file fp = fopen("test");
    if( fp==ferr ) {
        print("open file error. %e\n");
        return;
    };
    print("%s %s %d\n", fp, fp, fp);
    fclose(fp);
```

Example 6:

```
    string str = "name";
    int ret = query("show table %s", str);
    if( ret<0 ) {
        print("query failed. %e\n");
    };
```

Example 7:

```
int i = 1;
while( i<10 ) {
    if( i>5 ) {
        int i = 100;
        print("i:%d\n", i);
        break;
        print("shouldn't be here.\n");
    };
    i = i +1;
    print("the outside:%d\n", i);
};
print("end i:%d\n", i);
```