

Variables In FrameScript Subroutines

v1.1

2001/03/06

Philip Sharman

<http://members.home.com/philip.sharman/FrameScript.htm>

These are some notes on how variables are passed to and from subroutines in FrameScript.

I wrote them because I don't find this as intuitive in FrameScript as it is in some other languages. (As far as I know everything said here is correct, but I make no guarantees.)

Note: There seems to be a bug in FrameScript. Sometimes, if you copy a script into the Script Window and run it from there, the output is not what you get if you run the script via the "Run" menu command. I've seen it happen, but I can't reproduce it. So, if the output you get doesn't match what is shown here, try running the script from outside the Script Window.

1. The Normal Case

Variables are passed to subroutines and passed back from subroutines by enclosing values in parentheses. (See the *FrameScript Reference Manual* under "Run Command" and "Sub SubName".)

The variables do not have to be defined before you use them in the "run" command.

For example, this script:

```
run subOne A(3) B(4) returns C(x);
write console 'x = ' + x; // 34

////////////////////////////////////
sub subOne using A B returns C
  set C = 10*A + B;
endSub
```

produces this output:

```
x = 34
```

2. Passing Variables Implicitly

If you don't pass variables in parentheses, then the subroutine uses whatever values those variables already have.

The script:

```
set A = 3;
set B = 4;
set X = 100;
set Y = 200;

// All these produce the same output...
run subOne A B returns C(x1);
write console 'x1 = ' + x1; // 34

run subOne B A returns C(x2);
write console 'x2 = ' + x2; // 34

run subOne X Y returns C(x3);
write console 'x3 = ' + x3; //
```

```

run subOne returns C(x4);
write console 'x4 = ' + x4; // 34

////////////////////////////////////
sub subOne using A B returns C
  set C = 10*A + B;
endSub

```

produces:

```

x1 = 34
x2 = 34
x3 = 34
x4 = 34

```

3. Passing Values Explicitly

If you pass values in parentheses, then the subroutine uses those values. Those assignments are temporary, used only in the subroutine.

The script:

```

set A = 3;
set B = 4;

run subOne A(5) B(6) returns C(x);
write console 'x = ' + x; // 56

run subOne B(5) A(6) returns C(y);
write console 'y = ' + y; // 65

write console 'A = ' + A; // 3
write console 'B = ' + B; // 4

////////////////////////////////////
sub subOne using A B returns C
  set C = 10*A + B;
  write console 'In the subroutine, A = ' + A + ', B = ' + B
endSub

```

produces:

```

In the subroutine, A = 5, B = 6
x = 56
In the subroutine, A = 6, B = 5
y = 65
A = 3
B = 4

```

4. “Using” Is Optional

The word “using” appears to be optional in “sub” line. In fact, you don’t even seem to need to list the arguments on the “sub” line

This script:

```

run subOne A(3) B(4) returns C(x);
write console 'x = ' + x; // 34

run subTwo A(5) B(6) returns C(y);

```

```

write console 'y = ' + y; // 56

////////////////////////////////////
sub subOne A B returns C
  set C = 10*A + B;
endSub
////////////////////////////////////
sub subTwo returns C
  set C = 10*A + B;
endSub

```

produces:

```

x = 34
y = 56

```

5. Passing Values Implicitly and Explicitly

If you wish, you can explicitly assign one of the arguments but not the other.

This script:

```

set A = 3;
set B = 4;

run subOne A(5) returns C(x);
write console 'x = ' + x; // 54

////////////////////////////////////
sub subOne using A B returns C
  set C = 10*A + B;
  write console 'In the subroutine, A = ' + A + ', B = ' + B
endSub

```

produces:

```

In the subroutine, A = 5, B = 4
x = 54

```

6. The Return Parameter

The argument used to receive the return variable (e.g. 'C') must match the name used in the subroutine or else nothing gets passed back.

This script:

```

// This works
run subOne A(5) B(6) returns C(x);
write console 'x = ' + x; // 56

// This does not work
run subOne using A(5) B(6) returns D(y);
write console 'y = ' + y; // 0

////////////////////////////////////
sub subOne using A B returns C
  set C = 10*A + B;
endSub

```

produces:

```
x = 56
y = 0
```

7. More Than One Return Parameter

I had thought that you can only one return one parameter, but Michael Mueller-Hillebrand showed me how to get FrameScript to return multiple parameters.

This script:

```
run subOne A(5) B(6) returns C(x) returns D(y);
write console 'x = ' + x; // 56 // Okay
write console 'y = ' + y; // 30 // Okay

////////////////////////////////////
sub subOne using A B returns C D
  set C = 10*A + B;
  set D = A*B;
endSub
```

produces:

```
x = 56
y = 30
```

8. Local Versus Global Variables

Changes made in the subroutine to the 'using' variables or 'returns' variable are local to the subroutine.

But in FrameScript v1, changes made in the subroutine to any other variables are global.

This script:

```
set A = 1;
set B = 2;
set C = 3;
set Z = 4;

run subOne A(5) B(6) returns C(x);
write console 'x = ' + x; // 56 // The caller change this
write console 'A = ' + A; // 1 // Subroutine does not change this
write console 'B = ' + B; // 2 // Subroutine does not change this
write console 'C = ' + C; // 3 // Subroutine does not change this
write console 'Z = ' + Z; // 300 // Subroutine does change this

////////////////////////////////////
sub subOne using A B returns C
  set C = 10*A + B; // This change is local
  set A = 100; // This change is local
  set B = 200; // This change is local
  set Z = 300; // This change affects the global variable
endSub
```

produces:

```
x = 56
A = 1
B = 2
C = 3
Z = 300
```

In FrameScript v2, the new “local” command can be used to create variables that are local to a subroutine.

The script:

```
set y = 1;
set z = 2;

run subOne A(5) B(6) returns C(x);
write console 'x = ' + x; // 56
write console 'y = ' + y; // 10 // Subroutine changes this
write console 'z = ' + z; // 2 // Subroutine does nochange this

////////////////////////////////////
sub subOne using A B returns C
  local z; // Declares z to be a local variable
  set y = 10; // This changes the global variable
  set z = 10; // This change is local
  set C = z*A + B;
endSub
```

produces:

```
x = 56
y = 10
z = 2
```