

# Flow Control in G7

September 10, 2006

This document describes the flow control capabilities of G7. These few commands allow the user to write compact scripts with less repetition, and to perform complex tasks with greater ease.

This document provides details for flow control in the G7 scripting language. In addition, sample code with comments is provided below. A zipped file is available that contains G7, this documentation, and demonstration files similar to those below.

Use of these commands requires that G7, Version 7.3689 or later, be installed on your computer. These capabilities are new for G7, and hence more testing and debugging is required. Please report suggestions or problems.

Ron Horst

# Example 1

Demo file should be installed in the ifelse directory.

```
ic Testing if-else functions
ic -----

function TEST{
  if( NARGS == 0 ){
    ic Zero arguments were passed to the function TEST.
  }
  else if( NARGS == 1 ){
    ic The argument %1 was passed to the function TEST.
  }
  else{
    ic Two arguments, %1 and %2, were passed to the function TEST.
  }
}

TEST
TEST ONE_ARGUMENT
TEST FIRST_ARGUMENT SECOND_ARGUMENT

if( D != N ){
  ic String inequality test.
}
if( D <= N ){
  ic String less than or equal to comparisons.
}
if( D < N ){
  ic String less than comparisons.
}
if( D >= A ){
  ic String greater than or equal to comparisons.
}
if( D > A ){
  ic String greater than comparisons.
}

if( -1 < -0.5 ){
  ic Evaluation with negatives.
}

if( 1 && 2 < 3 && 3 <= 4 || 0 ){
  ic Multiple evaluations with OR and AND.
}

if( 0 ) {
  ic Single eval of integer.
}
else if( 0 ){
  ic Else-if evaluation of integer.
}
else{
  ic Else.
}

if( 1 < 2 ){
  ic Integer comparison.
}
```

Example continues on next page.

```

if( 1 >= 1 ){
    ic Greater than or equals works!
}

if( 1 != 2 ){
    ic Integer inequality test.
}

if( A != B ){
    ic String equality test.
}
else if( a > A ){
    ic Inequality test, with capital and lowercase letters.
}
if( A != B && 3 > 2 ){
    ic Multiple comparisons.
}

do{
    if( %1 == 3 ){
        ic Continuing on Iteration %1
        continue
    }
    if( %1 == 3 ){
        ic Printing on iteration 3
    }
    if( %1 == 5 ){
        ic Breaking on Iteration 5
        break
    }
    if( %1 == 6 ){
        ic Should not have gotten to %1.  Escaping....
        return ESC
    }
    if( %1 == 7 ){
        ic Certainly should not have gotten to %1.  Returning ERR....
        return ERR
    }
    if( %1 > 7 ){
        ic Things are not OK, but returning OK anyway....
        return OK
    }
    ic Iteration %1
}(1-10)

if(1){
    do{
        do{
            ic Outer loop %1, inner loop %2.
        }(1-2)
    }(1-3)
}

ic -----

```

## Syntax

### break

This command halts execution of a do loop or an add file, but processing continues of the script after the do loop or of the script that called the add file.

Examples:

```
do{
  if( %1 == 2 ){ break }
  else{ ic This is iteration %1 }
}(1-3)
```

### continue

This command causes the current iteration of a do loop to end, and the next iteration to begin.

Examples:

```
do{
  if( %1 == 2 ){ continue }
  else{ ic This is iteration %1 }
}(1-3)
```

```
if <(> [comparisons] <){> [...] <>>
[ else if <(> [comparisons] <){> [...] <>> ]
[ else <{> [...] <>> ]
```

The `if` command evaluates a series of logical expressions. If the arguments are true, then the following block of code, contained in a set of brackets, is executed. If the argument is false, then `else if` statements are processed in the same way. If all `if` and `else if` arguments are false, then the code following the final `else` command is processed.

Three types of comparisons are legal. First, a number may be compared to a second number. Second, a single number may be given and implicitly compared to zero. Finally, a string may be compared to a second string, with the comparison based on the lexicographical ordering.

Valid comparisons between two strings or between two numbers are

<	is LESS THAN
<=	is LESS THAN OR EQUAL TO
==	is EQUALS
>=	is GREATER THAN OR EQUAL TO
>	is GREATER THAN
!=	is NOT EQUALS

### and

! is the logical negation operator. If given before a number, then the result is FALSE if the number is nonzero and the result is TRUE if the number is zero. If given before a

set of parentheses, then the result is FALSE if the expression within the parentheses is TRUE, and the result is TRUE if the expression within the parentheses is FALSE.

Multiple comparisons may be performed, with the following operators joining them

|| Logical OR, which returns TRUE if either the left hand side or the right hand side is TRUE.

&& Logical AND, which returns TRUE if both the left hand side and the right hand sides are TRUE, and returns FALSE otherwise.

The following list of keywords may be used

NARGS

The number of arguments passed to a function or to an add file.

Comparisons may be evaluated as a group by surrounding them with parentheses. Blocks of code following the arguments must be surrounded by brackets {}.

Examples:

```
if( -1 < -0.5 ){          ic Evaluation with negatives works!          }
if( 2 < 3 && 3 <= 4 ){     ic Multiple evaluations with AND work!      }
if( 2 < 3 || 3 <= 4 ){    ic Multiple evaluations with OR work!       }
if( 1 == 1 ){             ic Equals works!                             }
if( 1 != 2 ){             ic Not equals works!                         }
if( 1 ){                  ic Single evaluation works!                  }
else if( !0 ){            ic Else if and negation work!                 }
else{                     ic Else works!                               }

if( D != N ){             ic String inequality comparisons work!        }
if( D <= N ){             ic String less than or equal to comparisons work! }
if( Doug == doug ){       ic String equality comparisons work!         }
if( D != d && 3 > 2 ){     ic Multiple comparisons strings and numbers!  }

do{
  if( %1 == 3 ){
    ic Continuing on Iteration %1
    continue
  }
  if( %1 == 5 ){
    ic Breaking on Iteration 5
    break
  }
  if( %1 > 5 ){ return ERR }
}( 1-10 )
```

return [arg]

This command halts execution of the G7 script. If G7 is in the midst of processing a do loop or an add file, then the arguments determine the actions of G7. Arguments may be given in upper or lower case.

Arguments:

OK The default argument. Exits a do loop or add file, but processing continues.  
ESC Processing halts entirely, but no error messages are shown.  
ERR Processing halts entirely, with error messages displayed.

Examples:

```
return ERR
```

## Known Bugs and Limitations

At this time, the `if-else` commands cannot process arguments based on time series or vector data. They can, however, process arguments sent to a `do` loop or to an `add` file that contains the `if-else` statements. See the example above for details.