CSPro Language Reference

Commands in this reference have been updated for CSPro 4.1.001.

Table of Contents

Language Structure

Whitespace

The CSPro language ignores whitespace—spaces, tabs, and newlines—when compiling the code. Spacing code properly is a good way for the programmer to write clear code. These three lines of code are the same:

```
randomVal = random(1,100);

randomVal       =   random ( 1 , 100 ) ;

randomVal
=
random(1,100);
```

Single-Line Comments

A programmer can write comments for his or her benefit, or for the benefit of others. It is good practice to add many comments to code to make the intent of the code clearer. When the compiler reads the single-line comment characters, //, it will ignore the rest of the line. Typing Control + / in the logic editor will automatically comment the line of code.

```
area = 2 * 3.14 * radius; // circle area formula: 2 pi r
```

Multiple-Line Comments

Alternatively, a programmer can comment multiple lines of code with the multiple-line comment characters, { and }. These comments are very useful for having the compiler ignore a whole section of code. Unlike in some programming languages, multiple-line comments can be nested in CSPro.

```
{
do counter = 1 while counter <= totocc(POP_REC)
    errmsg("SEX(%d) = %d",counter,SEX(counter)); // debugging info
enddo;
}
```

PROC GLOBAL

set explicit / set implicit

CSPro can compile logic in two modes. If explicit is specified, all variables must be declared either globally in the PROC GLOBAL section of the code, or locally with a function or PROC. If implicit is specified and a variable name is given that has not been already created, CSPro will create the variable on-the-fly. It is highly recommended that code is compiled using explicit mode because programmers sometime misspell variable names and these errors are not discovered in implicit mode, potentially leading to unintended consequences. This option can also be specified in the Options menu, though that selection is only valid for the computer on which CSPro is currently installed.

```
PROC GLOBAL

set explicit; // recommended
set implicit;
```

numeric

This command creates a "working variable," a numeric variable that only exists while the program is running. It is not saved to any input or output data files. If defined in PROC GLOBAL, it can be accessed from anywhere in the code. This is called a global variable. On the other hand, a local variable is created if defined within a function or a PROC, and this variable can only be accessed from within that function or PROC. A variable can be given a value upon creation; if no value is given, it will be set to 0 by default.

```
PROC GLOBAL

numeric globalVariable; // will be 0 by default
numeric previousSex = 1; // assigning a default value other than 0

PROC AGE

    numeric localVariable1,localVariable2;
```

alpha

Using alpha create an alphanumeric (string) working variable that, like variables created using the numeric command, only exists during the program run. Alpha variables can only be declared in PROC GLOBAL, so all alphanumeric working variables has global scope. The size of the string, by default 16 characters, can be specified. CSPro will pad shorter strings with spaces until they fill the entire string.

```
PROC GLOBAL

alpha shortString; // will be 16 characters by default
alpha (10) shorterString = "Hello"; // will be "Hello     "
alpha (100) longerString;
```

array

CSPro can create one-, two-, or three-dimensional arrays. These must be defined in PROC GLOBAL, and the size of each dimension must be specified. Array access is one-based, and an index for each dimension must be specified to access a value in the array.

```
PROC GLOBAL

array oneDimArray(5);
array twoDimArray(5,10); // 5 rows, 10 columns
array threeDimArray(5,10,20); // 5 rows, 10 columns, 20 layers

PROC VAL

    oneDimArray(3) = 1;
    twoDimArray(4,8) = 100;
    threeDimArray(3,1,17) = 1000;
```

Advanced note: Arrays in CSPro are actually a hybrid between being zero- and one- based. In the above example, oneDimArray actually has six elements, twoDimArray has 66 elements, and threeDimArray has 1,386 elements. It is suggested that programmers treat arrays as if they were one-based, though the element at index zero can be used if desired.

Arrays can also be created that contain strings, not numeric variables. It is also possible to specify the initial values of the array while defining the parameters of the array. A fill operator, …, will copy the previously specified values over and over until the array is filled.

```
PROC GLOBAL

array alpha smallStringArray(5); // 5 16-character strings
array alpha (100) stringArray(5) // 5 100-character strings

array ticTacToe(3,3)    =   1   0   2
                            2   1   0
                            0   1   1;

array fillArray(9) = 1 2 3 ...; // will be: 1 2 3 1 2 3 1 2 3
```

In batch programs, the contents of an array can be saved to a file after the running of a program. The next time the program run, the values of the array will be loaded from the file. This feature is useful when creating hotdecks because the values that populate the hotdeck will be carried over from one run of the program to the next. By default CSPro will create one saved array file (.sva) for an application, but a programmer can indicate that CSPro should maintain a saved array file for each input data file passed to the program.

```
PROC GLOBAL

array incomeHD(2,15,20) save; // sex x education x age

// use a different saved array file for each input file
set array save(filename);
```

When using DeckArrays (covered later in this guide), at least one of the dimensions must be the name of a value set instead of a number. The number of elements in the value set determines the size of the array.

```
PROC GLOBAL

array incomeHD(VS_SEX,VS_EDUCATION,VS_AGE5YEARS) save;
```

file

To use external files, a file handler must be specified. This file handler allows CSPro to read from or write to multiple external files. The file that the handler points to can be specified in the program's PFF file, or it can be specified in the program's logic.

```
PROC GLOBAL

file inputFile,csvOutputFile;
```

PROCs

CSPro code is executed upon certain triggers, and knowing in which PROC to place the statements, and upon what trigger, is important to writing successful CSPro code. There are several elements that can contain code:

- Application (this will end with _FF)
- Level
- Form / Record (data entry / batch)
- Group
- Field / Item (data entry / batch)

Each PROC can have multiple triggers, but they must be coded in the following order, which is also the order of execution:

- preproc
- onfocus
- onoccchange (only for groups)
- killfocus
- postproc

Most applications will use only preprocs and postprocs. Onfocus and killfocus exist for more advanced control over an application. While onfocus and killfocus work in batch applications, putting code in the onfocus is no different than placing it in preproc, and killfocus code could be placed in postproc. Onoccchange is rarely used. In none of the trigger keywords is given, the statements will be executed as if they were a postproc.

preproc

In a data entry application, code in a preproc will be executed as the cursors moves forward into the element, but not when the keyer moves backwards on the form. Code in a preproc is executed before any of the other code triggers are executed.

onfocus

After any code in a preproc is executed (if applicable), the code in onfocus is executed. Code in onfocus is similar to code in a preproc, except that in a data entry application, code in onfocus will be executed whenever the cursors moves to the element, whether the cursors is moving forward or backwards. Code in an onfocus is guaranteed to run before a keyer can enter a value for the element, which is not the case with preproc.

killfocus

When a keyer tries to leave a field or other element, the element's killfocus is executed. This can happen in the standard way after the keyer enters a value. It can also happen if the keyer tries to move backwards in the form, or uses the mouse to move to another element.

postproc

Code in a postproc is executed whenever a keyer completes entry of a value in a field, after killfocus is executed (if applicable), as the cursor moves forward in the form. Postproc code will not be executed if the user moves backwards in the form, or clicks off a field with the mouse in operator-controlled mode.

onoccchange

This trigger can be used with groups (repeating records or repeating items) but it is rarely used in CSPro code. The code in this trigger will be executed whenever the occurrence number of a group changes. For instance, if a horizontal roster is on the form, the code will be executed every time the cursor moves to a new row of the roster.

```
PROC MILES

preproc

    numeric lat1,long1,lat2,long2;

    // lookup airport codes (not shown)

    numeric minDistance =
            gps(distance,lat1,long1,lat2,long2) * 0.621371192;

    MILES = minDistance;

postproc

    if MILES < minDistance then
        errmsg("Flight distance is too short");
        reenter;
    endif;

    TOTALMILES = sum(MILES);
```

```
PROC PASSWORD

// hide the password unless the keyer is on that field

onfocus

    set attributes($) visible;

killfocus

    set attributes($) hidden;
```

Conditional Statements

if / elseif / else / endif

One or several conditional statements can be linked together to control what logic statements are executed. Each if or elseif statement is followed by a condition and the word “then.” An else statement, which is executed if none of the previously linked if and elseif statements were executed, does not need the “then” statement. An if / elseif / else chain must be terminated with endif. It is a good idea to use good indentation when writing conditional statements (especially nested conditional statements) so that it is obvious which if an elseif or else refer to.

```
PROC MARST

    if MARST = 1 then
        // ...

    elseif MARST = 2 then

        if AGE < 15 then
            errmsg("Age too young to be married.");
        endif;

    elseif MARST = 3 then
        // ...

    else
        errmsg("Invalid marital status value.");

    endif;
```

Relational Operators

All relational operators evaluate to 1 if the statement is true, and 0 if the statement if false. Unlike some programming languages, CSPro does not have different operators for assignment and for checks of equality. This means that assignments cannot be made as part of a conditional statement.

- Equals: =
- Does not equal: <>
- Less than: <
- Less than or equal to: <=
- Greater than: >
- Greater than or equal to: >=

```
if AGE >= 13 and AGE <= 19 then
    errmsg("Person is a teenager");
endif;

// this shows using an assignment and a check for equality together
numeric isBaby = AGE = 0; // isBaby will be 1 or 0
```

in

The in operator gives as programmer an easy way to check whether a value is within a range or in a list of several discrete numbers or ranges.

```
if AGE in 12:49 then
    errmsg("Fertility data expected");
endif;

if not TRIBE in 101:104,201:209,301:302,999 then
    errmsg("Invalid tribe");
endif;
```

<=> if and only if

An if and only if operator exists in CSPro, but it is rarely used and tends to be a confusing operator for programmers. Code written with the if and only if operator can be written more clearly, though less succinctly, using other operators. It is equivalent to the iff or xnor operator in other programming languages, and returns the opposite truth table as the exclusive or, xor.

Logical Operators

Combining multiple conditional statements into one condition is a useful feature in programming. The order of precedence matters in CSPro so it is important to scrutinize any statement with multiple conditions to ensure that it is being evaluated properly. Parentheses can be used to override the default precedence. All of the logical operators have a word and a character version:

- Negation: not, !
- Conjunction: and, &
- Disjunction: or, |

With conjunctions, CSPro evaluates each condition until one is false. On the other hand, with disjunctions, CSPro evaluates each condition until one is true. This means that in both cases, all parameters are not necessarily checked.

```
// if ptrSpouse = 0, the conjuction will end and CSPro will
// not try to evaluate SEX(ptrSpouse), which was invalid at SEX(0)
if ptrSpouse and SEX(ptrSpouse) = SEX(1) then
    errmsg("Spouse and head are of the same sex");
endif;

if not invalueset(TRIBE) or TRIBE = 0 then
    errmsg("Tribe was invalid or 0");
endif;
```

Order of Operator Precedence

With operators at the same precedence, the highest precedence is given from left to right. Use parenthesis to alter the order of precedence.

Highest Precedence	^
	* / %
	+ -
	= < > <= >= <> in
	not
	and
	or
Lowest Precedence	<=>

Loops

while

The while loop will continue running until the looping condition evaluates as false. If using a counter to loop through a record or an array, the programmer must change the value of the counter for each iteration of the loop.

```
numeric counter = 2;

while counter <= totocc(POP_REC) do

    if AGE(counter) > AGE(1) then
        errmsg("Head of household is not the oldest");
    endif;

    inc(counter);

enddo;
```

do

A do loop is similar to a while loop but gives the programmer added functionality. One feature of a do loop is that the loop can automatically increment (or decrement) the counter for the loop by a given value. If a given value is not specified, the loop will automatically increment the counter by 1 after each iteration through the loop. Another feature is that instead of making the loop condition a positive one, a negative condition can be given. In this case, the loop will run until condition the condition becomes true.

```
numeric counter;

do counter = 2 while counter <= totocc(POP_REC)

    if AGE(counter) > AGE(1) then
        errmsg("Head of household is not the oldest");
    endif;

    // no need for inc(counter);

enddo;

do counter = totocc(POP_REC) until counter = 1 by -1

    if AGE(counter) > AGE(1) then
        errmsg("Head of household is not the oldest");
    endif;

enddo;
```

for

A for loop is used to easily loop through a group (a repeating record or item). Within the loop, a subscript is not needed to refer to the current element, which can be handy if doing a lot of processing within the loop.

```
numeric counter;

for counter in POP_REC

    if counter > 1 and AGE > AGE(1) then
        errmsg("Head of household is not the oldest");
    endif;

endfor;
```

next

The next statement terminates processing the current iteration of the loop and returns to the loop condition to determine whether to run the loop again.

```
for counter in POP_REC

    if SEX = 1 then
        next;
    endif;

    // process women here

endfor;
```

break

To terminate a loop early before all iterations of the loop have run, the break statement can be coded. As soon as the statement is executed, the program will pass on to the code following the loop. In a nested loop, break quits out of the innermost loop.

```
numeric ptrSpouse;

for counter in POP_REC

    if RELATIONSHIP = 2 then
        ptrSpouse = counter;
        break;
    endif;

endfor;
```

User-Defined Functions

function

In the PROC GLOBAL section of code, a programmer can specify user-defined functions that the programmer intends to use multiple times throughout the code. Generalizing code into a function is good programming practice, as modifications to code only have to made in one place and the size of code is reduced. A function will return a number by default, but can also return a string. To assign a return value, set the name of the function to value.

```
function numberDaysSince2000() // a numeric return value

    numeric todaysDate = sysdate("YYYYMMDD");

    numberDaysSince2000 = datediff(20000101,todaysDate,"d");

end;

alpha (3) personTitle;

function alpha (50) formatName() // a string return value

    if      SEX = 1 then                personTitle = "Mr";
    elseif  SEX = 2 and MARST = 1 then  personTitle = "Ms";
    else                                personTitle = "Mrs";
    endif;

    formatName = maketext("%s. %s %s",
                          strip(personTitle),strip(FNAME),LNAME);

end;
```

exit

To terminate processing within a function early, use the exit command. This same command can be used to stop processing a PROC, moving to the next PROC (or trigger within the same PROC).

```
function findFertileWoman()

    numeric counter;

    do counter = 1 while counter <= totocc(POP_REC)

        if SEX(counter) = 2 and AGE(counter) in 12:49 then
            findFertileWoman = counter;
            exit;
        endif;

    enddo;

    findFertileWoman = 0; // count not find anyone

end;
```

Function Parameters

Functions without parameters can be useful, though adding parameters increases the power of user-defined functions. A parameter is a temporary variable that exists only for the life of the function. There are three kinds of parameters to functions: numeric variables, strings, and arrays. If a numeric variable or string is modified within the function, these functions do not affect the values of the passed parameters. That is, the parameters are passed by value.

```
function isLeapYear(year) // year is the passed numeric variable

    if ( year % 4 = 0 and year % 100 <> 0 ) or year % 400 = 0 then
        isLeapYear = 1;

    else
        isLeapYear = 0;

    endif;

end;

function countVowels(alpha (100) sourceString)

    sourceString = tolower(sourceString);

    numeric numVowels,startSearchPos = 1,lastSearchPos;

    while startSearchPos do

        startSearchPos =
                    poschar("aeiou",sourceString[lastSearchPos + 1]);

        if startSearchPos then
            inc(numVowels);
            inc(lastSearchPos,startSearchPos);
        endif;

    enddo;

    countVowels = numVowels;

end;


PROC TEST

    if isLeapYear(YOB) then
        errmsg("Person born on a leap year");
    endif;

    errmsg("%s's name consists of %d vowels",NAME,countVowels(NAME));
```

tblrow / tblcol / tbllay

Arrays are passed to functions by reference, meaning that any changes to the arrays within the function affect the source array. The number of dimensions of an array must be specified when creating the function, but not the size of each dimension. To access the size of each dimension, use the tblrow, tblcol, and tbllay functions for the sizes of the first, second, and third dimensions, respectively.

```
function arrayMean(array oneDimArray)

    numeric counter,arraySum;

    do counter = 1 while counter <= tblrow(oneDimArray)
        inc(arraySum,oneDimArray(counter));
    enddo;

    arrayMean = arraySum / tblrow(oneDimArray);

end;

function geometricMean(array threeDimArray(,,))

    numeric x,y,z,numElements,product = 1;

    do x = 1 while x <= tblrow(threeDimArray)
        do y = 1 while y <= tblcol(threeDimArray)
            do z = 1 while z <= tbllay(threeDimArray)
                product = product * threeDimArray(x,y,z);
            enddo;
        enddo;
    enddo;

    numElements = tblrow(threeDimArray) *
                  tblcol(threeDimArray) * tbllay(threeDimArray);

    geometricMean = product ^ ( 1 / numElements );

end;

array smallArray(6) = 11 22 33 44 55 66;
array largerArray(13) = 0 1 1 2 3 5 8 13 21 34 55 89 144;
array eightValues(2,2,2) = 1 2 4 8 16 32 64 128;

PROC TEST

errmsg("Mean of small array: %0.1f",arrayMean(smallArray)); // 38.5
errmsg("Mean of larger array: %0.1f",arrayMean(largerArray)); // 28.9
errmsg("Geometric mean: %0.1f",geometricMean(eightValues)); // 11.3
```

OnKey

In a data entry application, there is a special function, OnKey, that can be overridden by a programmer. If the function is overridden, CSPro will call the function every time that a keyer types a keystroke. The return value of the function will be the code of the keystroke, which may be modified by the function. In

OnStop

sdf