Empowering CSPro

October 2010
by Guido Pieraccini
g.pieraccini@gmail.com
User-Defined Functions

• User-defined functions are coded in the declaration portion (PROC GLOBAL) of an application. Once defined, they can be used anywhere in an application.

• User-defined functions are used to perform operations that are used in several different places in an application.

• Functions are of the form: 
  Return-value = function-name(parameter-list)
• Functions must include a parameter-list which can vary depending on the function call’s requirements. This list may be null or it may contain one or more parameters.

• Each parameter specifies a variable that is used by the statements within the function. These variables are local to the function. That is, if a variable is passed as a parameter, its value in the rest of the application will not be changed by any action within the function.

• All other variables used in a function are global in scope: they can be changed anywhere in the application, including inside the function.
A user-defined function:

• return a single value;
• can contain CSPro statements and functions, as well as other user-defined functions. If no return value is assigned to the function, a default value is returned;
• cannot be recursive (i.e., they cannot call themselves), though they can call other functions either user-defined or CSPro Built-in function.

The **Function** statement allows the creation of a user-defined function.
Function Statement

Format:

• `function function-name([p-1,p-2...,p-n]);`
  
  `statements;`
  
  `function-name = numeric-exp;`

  `end;`

• `return-value = function-name([p-1,p-2...,p-n]);`

[ ] indicates that this part is optional.
• Numeric or alpha expressions can be passed to a user-defined function as parameters. Variables mentioned in the parameter list will by default be considered numeric.

• To specify an alphanumeric parameter, you must place the keyword *alpha* before the parameter. By default, the length of the alphanumeric parameters is 16 characters. To specify a different length, place *alpha (length)* before the parameter name.
• Be aware that the names used in the parameter list should not be the same as names defined in any dictionary associated with the application, nor should the parameter’s names be the same as variables defined in a numeric or alpha declaration statement.

• Functions return a numeric value unless different specified. To specify an alpha return-value, you must place the keyword *alpha* before the name of the function.

• To specify the return-value, assign a value to the name of the function. If no return value is assigned to the function, a DEFAULT value is returned.
Example 1: Create a string concatenating the case ID
Parameter: Region, District, Ward, EA, HH
Returns: a string with the concatenated ID

- alpha(2) dStr1;
- alpha(1) dStr2;
- alpha(3) dStr3, dStr4;
- alpha(4) dStr5;

Function alpha (13) ConcatID (MyPar1, MyPar2, MyPar3, MyPar4, MyPar5);
    dStr1=edit ("99", MyPar1);
    dStr2=edit ("9", MyPar2);
    dStr3=edit ("999", MyPar3);
    dStr4=edit ("999", MyPar4);
    dStr5=edit ("9999", MyPar5);
    ConcatID= concat (dStr1, dStr2, dStr3, dStr4, dStr5);
end;
• Or in alternative:
  Function alpha (13) ConcatID (alpha (2) MyPar1, alpha (1) MyPar2, alpha (3) MyPar3, alpha (3) MyPar4, alpha (4) MyPar5);
    ConcatID= concat (MyPar1, MyPar2, MyPar3, MyPar4, MyPar5);
  end;

• Note that the main difference between the two codes is that in the first case the parameter required are numbers while in the second case they are required to be strings.

• Note also that in both the cases the returned value it is a string of 13 characters
External Dictionary

• An external dictionary has the same components of a basic dictionary but the purpose of usage is different. External dictionaries are used mainly for LOOKUP files, for check the boundaries of a specific value or for storing some additional data as the other specify values.

• In the NPS application they were used for checking the Geographical codes, checking the GPS codes and the consumptions values, to store the Overridden Skips.
The `writecase` function writes a case from memory to an external dictionary. It can be used to update existing cases or to write new ones.

- The `ext-dict-name` must be supplied. It is the internal name of the dictionary defined in the application for the external file.
- The optional `var-list` defines the case identifiers in the external file. The `writecase` function concatenates the variables specified in `var-list` to form a string whose length must be the same as the length of the case identifier in the external dictionary. All the variables in the `var-list` must exist in the external dictionary.
- The function returns a value 1 (true) if the case was written successfully, 0 (false) otherwise.
Example 2: Define an override skip for the current field

Parameter: none
Returns: 1 if created, 0 if not

- function create_override_skip()
  overskip_id = concatID(ID_01, ID_02, ID_03, ID_04, ID_05);
  field_name = getsymbol();
  field_occr = curocc();
  create_override_skip = writecase(override, overskip_id, field_name, field_occr);
end;

- Note that create_override_skip function required no parameters. The current field name is read with the getsymbol function while the current occurrence with the curocc functions.

- Note that the user-defined function ConcatID it is called from inside create_override_skip.
Loadcase Function

\[ b = \text{loadcase}(\text{ext-dict-name}[, \text{var-list}]); \]

- The loadcase function reads a specified case from an external file into memory. Once the case is loaded, all the variables defined in the corresponding external dictionary are available for use.
- The "ext-dict-name" must be supplied. It is the internal name of the dictionary defined in the application for the external file.
- The optional "var-list" specifies the list of variables that will identify the case to load from the external file. This process is similar to matching files on the basis of key variables in a database.
- The function returns a value 1 (true) if the case is loaded successfully, 0 (false) otherwise.
Example 3: search the current field in the override list

Parameter: none
Returns: 1 if found, 0 if not

- function find_override_skip()
  overskip_id=concatID(ID_01, ID_02, ID_03, ID_04, ID_05);
  field_name = getsymbol();
  field_occr = curocc();
  find_override_skip=loadcase(override, overskip_id, field_name, field_occr);
end;

- Note that find_override_skip function required no parameters. The current field name is read with the getsymbol function while the current occurrence with the curocc functions.

- Note that the user-defined function ConcatID it is called from inside find_override_skip.
Delcase Function

\[ b = \text{delcase}(\text{ext-dict-name}[, \text{var-list}]); \]

- The delcase function marks a case for deleting in the external file associated with the ext-dict-name. The case whose identifiers match var-list is the case that is marked as deleted. A deleted case is marked by a tilde ‘~’ in the first character of each record in the case.
- The ext-dict-name must be supplied. It is the dictionary name defined in the data dictionary for the external file.
- The optional var-list defines the case identifiers in the external file. The delcase function concatenates the variables specified in var-list to form a string whose length must be the same as the length of the case identifier in the external dictionary. All variables in the var-list must exist in the external dictionary.
- The function returns a value 1 (true) if the case is successfully marked for deletion, 0 (false) otherwise.
Example 4: delete the override skip for the current field

Parameter: none

Returns: 1 if deleted, 0 if not

- function delete_override_skip()
  overskip_id=concatID(ID_01, ID_02, ID_03, ID_04, ID_05);
  field_name = getsymbol();
  field_occr = curocc();
  delete_override_skip=delcase(override, overskip_id, field_name, field_occr);
end;

- Note that the user-defined function ConcatID it is called from inside delete_override_skip.

- Note that delete_override_skip function required no parameters. The current field name is read with the getsymbol function while the current occurrence with the curocc functions.
ONKEY FUNCTION

- **Function OnKey(key-value);**
- The onkey global function allows you to trap keystrokes in order to perform special actions or to change the action of the key (remap keys). This function must be placed in the Global Procedure.
- If an onkey global function is coded, every keystroke the operator types is sent to the onkey function for processing. If the onkey function returns a value, then the return value is processed by the field as the keystroke.
- The key value is a number code identifying what key was pressed on the keyboard. Its value can be used within the function.
- The onkey global function returns an integer value. The value should be either the value of the pressed key or a substituted key value or zero (0) to indicate that the key is to be ignored.
Example 5: Manage Override Skips

Parameter: ascii value

Returns: none

Function OnKey(ascii);
FlagOverride=0;
if ascii=114 then
    {*** Manage Override Skips (F3) ***}
    if find_ovation_skip()=1 then
        k=0;
        do until k in 1:2
            k=accept("Do want to Remove the Overridden Skip?", "No", "Yes");
        enddo;
        if k=2 then
            delete_override_skip();
            FlagOverride=0;
        endif;
    Onkey=0;
else
  k=0;
  do until k in 1:2
    k=accept("Want to Override this Skip?", "No", "Yes");
  enddo;
  if k=2 then
    create_ovverride_skip();
    FlagOverride=1;
    OnKey=13;
  else
    Onkey=0;
  endif;
endif;
else
  if find_ovverride_skip()=1 then
    FlagOverride=1;
  endif;
endif;
end;
i = demode();

• The **demode** function is often used to limit the execution of certain statements to a specific mode. For example, a variable may need initialization when the operator invokes **add** or **verify** mode, but can be left unaltered for **modify** mode.

• There are three data entry operator modes:
  – **add**, to input new cases; the demode function returns a '1'
  – **modify**, to change cases that have already been entered; the demode function returns a '2'.
  – **verify**, to reenter the cases as a check for differences between the first and second entry; the demode function returns a '3'.

Example 6: Defining the Working Mode

Parameter: none
Returns: 1-4

Function WorkingMode();
    if demode()=1 then
        {Data Entry in Add Mode}
        WorkingMode=1;
    elseif demode()=2 then
        {Data Entry in Modify Mode}
        WorkingMode=2;
    elseif demode()=3 then
        {Data Entry in Verify Mode}
        WorkingMode=3;
    else
        {Batch Mode}
        WorkingMode=4;
    endif;
end;
Example 7: SkipTo with override management

Parameter: Skip Target
Returns: 0 skipped, 1 not skipped

- function SkipTo (alpha Skip_Target)
  
  SkipTo =0;
  if WorkMode in 1:3 then
    if FlagOverride=0 then
      skip to @skip_Target;
      SkipTo =1;
    endif;
  endif;
  elseif WorkMode =4 then
    skip to @skip_Target;
    SkipTo =1;
  endif;
  endif;
end;

- Usage: SkipTo (getsymbol(Filed_Name));
Example 8: SkipToNext with override management

Parameter: Skip Target
Returns: 0 skipped, 1 not skipped

- function SkipToNext (alpha Skip_Target)
  SkipToNext=0;
  if WorkMode in 1:43 then
    if FlagOverride=0 then
      skip to next @skip_Target;
      SkipToNext=1;
    endif;
  elseif WorkMode =4 then
    skip to next @skip_Target;
    SkipToNext=1;
  endif;
end;

- Usage: SkipToNext (getsymbol(Filed_Name));
Example 9: Message Box 1
Parameter: Text msg, Type of answer
Returns: Answer (1-3)

function MsgBox (alpha (150) MyStr, Type)
  if type=1 then
    do until k in 1:type
      k=accept(MyStr, "Yes");
    enddo;
  elseif type=2 then
    do until k in 1:type
      k=accept(MyStr, "No", "Yes");
    enddo;
  elseif type=3 then
    do until k in 1:type
      k=accept(MyStr, "No", "Yes", "Cancel");
    enddo;
  endif;
end;
**EXEC SYSTEM FUNCTION**

\[ b = \text{execsystem}(\alpha [\text{maximized} | \text{normal} | \text{minimized}], [\text{focus} | \text{nofocus}], [\text{wait} | \text{nowait}]); \]

- The `execsystem` function starts another windows application or process.
- The `\alpha`-exp is the name of the application or process to be started. Command line parameters may be included in this expression. If folders names or file names contain blanks, then quotation marks (")") must surround the names. In this case CSPro text strings must be surrounded by apostrophies (').
- `maximized`, `normal`, or `minimized` determines how the window for the new application is opened. If this parameter is not coded, the default is `normal`.
- `focus` or `nofocus` determines whether the new application will receive focus, or "be active," when it is started. If this parameter is not coded, the default is `focus`.
- `wait` or `nowait` determines whether the current application will wait until the new application is finished before control returns to it. If this parameter is not coded, the default is `nowait`.
- The function returns a logical value of 1 (true) if the new application is started successfully and 0 (false) otherwise.
Example 10: Message Box 2
Parameter: Text msg, Type of Box (vbscript )
Returns: User selection (vbscript)

- function MsgBox (alpha (150) MyStr, Type)
  Cmd=concat (' QMessageBox "", strip(Mystr), "" ', edit("Z", Type), ' "aw.txt"');
  execsystem (Cmd, minimized, focus, wait);
  setfile (FileAnswer, "aw.txt");
  fileread (FileAnswer, answer);
  setfile (FileAnswer, "xx.xxx");
  MsgBox=tonumber(answer);
end;

- Note that the external file MessageBox.exe should be in the main folder of the CSPro application or in the folder C:\Program Files\CSPro 4.0
- Note that the setfile/fileread function are used to get the user’s selection stored in the text file called “aw.txt”
Example 11: Input Box

Parameter: Text msg, Type of Box (vbscript)
Returns: User selection (vbscript)

- function alpha(64) InputBox (alpha (150) MyStr)
  Cmd=concat ('InputBox "', strip(Mystr), ' "aw.txt"');
  execsystem (Cmd, minimized, focus, wait);
  setfile (FileAnswer, "aw.txt");
  fileread (FileAnswer, answer);
  setfile (FileAnswer, "xx.xxx");
  InputBox=answer;
end;
- Note that the external file MessageBox.exe should be in the main folder of the CSPro application or in the folder C:\Program Files\CSPro 4.0
- Note that the setfile/fileread function are used to get the input value stored in the text file called “aw.txt”
Example 12: Create a Zip File Function
Parameter: Zip file path, Path of the File to compress
Returns: 1 if created, 0 if not

- Function FileZip (alpha(100) PathZip, alpha(100) PathFiles);
  Cmd=concat ('zip ''', strip(PathZip), ''' ''', strip(PathFiles), '''');
  execsystem (Cmd, minimized, focus, wait);
  res1=filesize (PathZip);
  if res1=-1 or res1=0 then
    FileZip=0;
  else
    FileZip=1;
  endif;
end;

- Note that the external file zip.exe should be in the main folder of
  the CSPro application or in the folder C:\Program Files\CSPro 4.0
Function FileUnzip (alpha(100) PathZip, alpha(100) PathDest);

    res1=filesize (PathZip);
    Cmd=concat ('unzip e "', strip(PathZip), '" -o"', strip(PathDest), '" -r -aoa');
    execsystem (Cmd, minimized, focus, wait);
    res2=filesize (PathZip);
    if res1<=res2 then
        FileUnzip=0;
    else
        FileUnzip=1;
    endif;
end;

Note that the external file unzip.exe should be in the main folder of the CSPro application or in the folder C:\Program Files\CSPro 4.0
Example 14: Replace a text in a txt file
Parameter: Search text, Replace Text, FilePath

- function ReplaceStr(alpha (30) search_text, alpha(30) replace_text, alpha(255) input_file)
  
  Cmd=concat('ReplaceText.exe "', strip(search_text), '" '', strip(replace_text), '" ', strip(input_file), '"');
  
  execsystem (Cmd, minimized, focus, wait);

end;

- Note that the external file ReplaceText.exe should be in the main folder of the CSPro application or in the folder C:\Program Files\CSPro 4.0
Example 15: Define the Main Application’s modality of execution when is run from a Menu Application

function GoInSystemCnt
    ReplaceStr("SystemControlled", "OperatorControlled", "MainApp.fmf");
end;

function GoInOperatorCnt
    ReplaceStr("OperatorControlled", "SystemControlled", "MainApp.fmf");
end;

function GoInBatch
    ReplaceStr("DataEntry", "Batch", "MainApp.ent");
end;

function GoInEntry
    ReplaceStr("Batch", "DataEntry", "MainApp.ent");
end;
Thank you