Subject: call_procedure with a dynamically created keywords list?
Posted by Robbie on Fri, 13 Oct 2006 00:37:10 GMT
View Forum Message <> Reply to Message

Folks,

I have long desired an equivalent of call_procedure which allows me to specify both input and output keywords in an array just as though you had used real keywords.

For example:
```
call_procedure_extra, 'dummy', ['A','B'], input_values, $
 ['X','Y'], output_values
```

Calls:
```
dummy, A=*input_values[0], B=*input_values[1], $
 X=*output_value[0], Y=*output_value[1]
```

The fundamental problem is that passing keywords by value is incompatible with arg_present().

I've finally managed to trick arg_present() with some smoke and mirrors, but I've found that arg_present() has some very odd behaviour. arg_present() is able detect if you have written a wrapper for a procedure. Indeed, this thwarts Dometz's efforts for a universal wrapper procedure, because there aint no fooling arg_present().

I apologise about the complexity and obscurity of the code below, but it's the only way I can demonstrate the problem.

```
;+
;<P>This is called by call_proedure_extra so that
; keywords appear as references</P>
;@private
;-
pro call_procedure_ref_extra, procedure_name, input_identifiers, $
 input_values, output_identifiers, output_values, _REF_EXTRA=ex
n_inputs = n_elements(input_identifiers)
n_outputs = n_elements(output_identifiers)
for i=0l,n_outputs-1 do $
 a = temporary(scope_varfetch(output_identifiers[i],/REF_EXTRA))
call_procedure, procedure_name, _EXTRA=ex
for i=0l,n_outputs-1 do begin
 if (n_elements(scope_varfetch($
  output_identifiers[i],/REF_EXTRA) gt 0)) then $
  *output_values[i] = $
  (scope_varfetch(output_identifiers[i],/REF_EXTRA))
endfor
```

```
end


;+
; <P>call_procedure_extra allows you to call a procedure,
; specifying the input and output keywords and values at runtime.
; The input and output keywords must be unique. Does not
; support having identical input and output keywords.</P>
; @param procedure_name {in}{required}{type=String}
; The procedure to call
; @param input_identifiers {in}{required}
; The keywords which are passed into the procedure
; @param input_values {in}{required}
; A pointer array of the value of each input keyword
; @param output_identifiers {in}{required}
; The keywords which are passed out of the procedure
; @param output_valyes {in}{out}{required}
; An allocated pointer array of the valye of each output keyword
;-
pro call_procedure_extra, procedure_name, input_identifiers, $
 input_values, output_identifiers, output_values
n_inputs = n_elements(input_identifiers)
n_outputs = n_elements(output_identifiers)
if ((n_outputs eq 0) and (n_outputs eq 0)) then begin
 call_procedure, procedure_name
 return
endif
if (n_inputs gt 0) then begin
 ex = create_struct(input_identifiers[0],*input_values[0])
 for i=1l,n_inputs-1 do $
  ex = create_struct(ex,input_identifiers[i],$
  *input_values[i])
 for i=0l,n_outputs-1 do $
  ex = create_struct(ex,output_identifiers[i],'')
endif else begin
 ex = create_struct(output_identifiers[0],'')
 for i=1l,n_outputs-1 do $
  ex = create_struct(ex,output_identifiers[i],'')
endelse
call_procedure_ref_extra, procedure_name, input_identifiers, $
 input_values, output_identifiers, output_values, _EXTRA=ex
end


;+
;<P>This is called by the test procedure</P>
;@private
;-
```

```
pro call_procedure_extra_dummy_encapsulated, _REF_EXTRA=ex
call_procedure_extra_dummy, _EXTRA=ex
end

pro call_procedure_extra_dummy, A=a, B=b, C=c, X=x, Y=y, Z=z
if (~arg_present(x)) then $
 print, 'X will not be passed back to the caller'
if (~arg_present(y)) then $
 print, 'Y will not be passed back to the caller'
if (~arg_present(z)) then $
 print, 'Z will not be passed back to the caller'
if (n_elements(a) gt 0) then x=a^1
if (n_elements(b) gt 0) then y=b^2
if (n_elements(c) gt 0) then z=c^3
end

;+
;<P>Test various combinations of keywords</P>
;@private
;-
pro call_procedure_extra_test
;
; arg_present seems to know the difference if I have created
; a wrapper procedure. Try the difference in commenting and
; uncommenting the different procedure names
;
procedure_name = 'call_procedure_extra_dummy_encapsulated'
;procedure_name = 'call_procedure_extra_dummy'
input_identifiers = ['A','B','C']
input_values = [ptr_new(1),ptr_new(2),ptr_new(3)]
output_identifiers = ['X','Y','Z']
output_values = ptrarr(3,/ALLOCATE_HEAP)
print, 'Testing with no keywords'
call_procedure_extra, procedure_name
for n=1,3 do begin
 print, '========================'
 print, 'Testing with ',n,' keywords'
 for i=0l,n-1 do print, 'Input: ', $
  input_identifiers[i], '= ', *input_values[i]
 call_procedure_extra, procedure_name, $
  input_identifiers[0:n-1], input_values[0:n-1],$
  output_identifiers[0:n-1], output_values[0:n-1]
 for i=0l,n-1 do print, 'Output: ', output_identifiers[i], $
  '= ', *output_values[i]
endfor
ptr_free, output_values
ptr_free, input_values
end
```