
Subject: Re: recursive tag_names

Posted by [davidf](#) on Sun, 22 Mar 1998 08:00:00 GMT

[View Forum Message](#) <> [Reply to Message](#)

Cathy (csaute3@alumni.umbc.edu) writes:

```
> Does anyone have a "recursive" tag_names like function. I would like to
> pass in a structure and get back a string array with members down to the
> bottom level. For example:
>
> input = {a:{m,n}, b:{r,s,t}, c, d, e}
>
> output = recursive_tag_names_function(input)
>
> print, output
>
> 'input.a.m', 'input.a.n', 'input.b.r', 'input.b.s', 'input.b.t',
> 'input.c', 'input.d', 'input.e'
```

It must have been a slow news day, or something, but for some reason this question intrigued me. I thought something like this might be useful for the object programming I am doing.

So here it is. There is a little main-level program below the two program modules that exercises the principle program, Get_Tags. If you just pass the structure to the program, you get all the fields with a "dot" in front of the names. For example,

```
tags = Get_Tags(struct)
```

returns:

```
.one
.one.two
.one.two.three
```

etc. If you want the full name back, pass a second positional parameter that is the root name of the structure. For example,

```
tags = Get_Tags(struct, 'struct')
```

returns:

```
struct.one
struct.one.two
struct.one.two.three
```

Note that the string vector that is actually returned from the Get_Tags function (line Tag_Names) is all UPPERCASE.

Cheers,

David

```
;-----  
Function All_Tags, structure, rootname  
  
; This is a function that recursively searches through  
; a structure tree, finding ALL of the structure's field names.  
; It returns a pointer to an array of pointers, each pointing  
; to the names of structure fields.  
  
IF N_Elements(rootname) EQ 0 THEN rootname = '.' ELSE $  
  rootname = StrUpCase(rootname) + '.'  
names = Tag_Names(structure)  
retValue = Ptr_New(rootname + names)  
  
; If any of the fields are structures, report them too.  
  
FOR j=0,N_Elements(names)-1 DO BEGIN  
  ok = Execute('s = Size(structure.' + names[j] + ')')  
  IF s[s[0]+1] EQ 8 THEN BEGIN  
    newrootname = rootname + names[j]  
    theseNames = Call_Function('All_Tags', $  
      structure.(j), newrootname)  
    retValue = [[retValue],[theseNames]]  
  ENDIF  
ENDFOR  
  
RETURN, retValue  
END  
;-----
```

```
FUNCTION Get_Tags, structure, rootname
```

```
; This function returns the names of all structure fields  
; in the structure as a string array. The names are given  
; as valid structure names from the root structure name,  
; which can be passed in along with the structure itself.
```

```
On_Error, 1
```

; Check parameters.

CASE N_Params() OF

0: BEGIN

Message, 'Structure argument is required.'

ENDCASE

1: BEGIN

rootname = "

s = Size(structure)

IF s[s[0]+1] NE 8 THEN \$

Message, 'Structure argument is required.'

ENDCASE

2: BEGIN

s = Size(structure)

IF s[s[0]+1] NE 8 THEN \$

Message, 'Structure argument is required.'

s = Size(rootname)

IF s[s[0]+1] NE 7 THEN \$

Message, 'Root Name parameter must be a STRING'

ENDCASE

ENDCASE

tags = All_Tags(structure, rootname)

; Extract and free the first pointer.

retval = [*tags[0,0]]

Ptr_Free, tags[0,0]

; Extract and free the the rest of the pointers.

s = Size(tags)

FOR j=1,s[2]-1 DO BEGIN

retval = [retval, *tags[0,j]]

Ptr_Free, tags[0,j]

ENDFOR

Ptr_Free, tags

; Return the structure names.

RETURN, retval

END

; Main-level program to exercise Get_Tags.

```
d = {dog:'spot', cat:'fuzzy'}
c = {spots:4, animals:d}
b = {fast:c, slow:-1}
a = {cars:b, pipeds:c, others:'man'}
tags = Get_Tags(a)
s = Size(tags)
For j=0,s[1]-1 Do Print, tags[j]
END
```

David Fanning, Ph.D.
Fanning Software Consulting
E-Mail: davidf@dfanning.com
Phone: 970-221-0438
Coyote's Guide to IDL Programming: <http://www.dfanning.com/>

Subject: Re: recursive tag_names
Posted by [Richard D. Hunt](#) on Fri, 27 Mar 1998 08:00:00 GMT
[View Forum Message](#) <> [Reply to Message](#)

I have written three routines to work with structures. The first is REMOVEELM.PRO which will take a structure and remove an element from it.

The second is REPLACEELM.PRO which will add or replace an element within a structure. The third is STRUCTELMNames.PRO will take a structure

and return a string array which lists every element of the structure. All of these routines will work on structures of structures. The only real side effect I have seen so far is that if you are using named structures the first two routines will return anonymous structures since you can't change a named structure.

Rich

```
;
;
; This function will remove an element of a structure.
;
; NOTE: This will return an anonymous structure since named a
;       named structure cannot be resized once it is defined.
;
; str: the structure you want to remove an element from.
; elm: the string name of the element to be removed.
```

```
; base: the "." separated string substructure of where to
; find the element to be removed. If this element has
; no "." in it then the particular element to be removed
; will be at the highest level.
```

```
; Example:
```

```
; IDL> str = {a:1, b:{c:2, d:{e:3, f:4}},g:5}
; IDL> nstr = RemoveElm(str, 'f', SubStructure='b.d')
; IDL> print, nstr
; { 1{ 2{ 3}} 5}
;
```

```
FUNCTION RemoveElm, str, elm, SubStructure=base
```

```
; Catch any errors.
```

```
Catch, error_status
```

```
IF error_status NE 0 THEN BEGIN
```

```
    print, "Couldn't remove element of structure."
```

```
    Return, str
```

```
ENDIF
```

```
; Get the names of the elements in str
```

```
names = StrLowerCase(Tag_Names(str))
```

```
element = StrLowerCase(elm)
```

```
IF N_Elements(base) NE 0 THEN BEGIN
```

```
    pos = StrLowerCase(StrTrim(Str_Sep(base, '.'), 2))
```

```
FOR i=0, N_Tags(str)-1 DO BEGIN
```

```
    type = Size(str.(i))
```

```
    IF type[N_Elements(type)-2] EQ 8 AND names[i] EQ pos[0] THEN
```

```
BEGIN
```

```
    ; Recurse into the structure.
```

```
    IF N_Elements(pos) GT 1 THEN BEGIN
```

```
        nb = String(Format='(100(A,.;"."))',pos[1:*])
```

```
        substr = RemoveElm(str.(i), elm, SubStructure=nb)
```

```
    ENDIF ELSE $
```

```
        substr = RemoveElm(str.(i), elm)
```

```
    ; Add the substructure to the structure.
```

```
    IF N_Elements(newstr) EQ 0 THEN $
```

```
        newstr = Create_Struct(names[i], substr) $
```

```
    ELSE $
```

```
        newstr = Create_Struct(newstr, names[i], substr)
```

```
    ENDIF ELSE BEGIN
```

```
    ; Add the substructure to the structure.
```

```
    IF N_Elements(newstr) EQ 0 THEN $
```

```
        newstr = Create_Struct(names[i], str.(i)) $
```

```

        ELSE $
            newstr = Create_Struct(newstr, names[i], str.(i))
        ENDELSE
    ENDFOR
ENDIF ELSE BEGIN
    FOR i=0, N_Tags(str)-1 DO BEGIN
        IF names[i] NE element THEN BEGIN
            IF N_Elements(newstr) EQ 0 THEN $
                newstr = Create_Struct(names[i], str.(i)) $
            ELSE $
                newstr = Create_Struct(newstr, names[i], str.(i))
            ENDIF
        ENDIF
    ENDFOR
ENDELSE

    Return, newstr
END

```

```

;
; This function will add or replace an element of a structure.
;
; NOTE: This will return an anonymous structure since named a
;       named structure cannot be resized once it is defined.
;
; str: the structure you want to remove an element from.
; elm: the string name of the element to be removed.
; val: the value to replace or add to the structure.
; base: the "." seperated string substructure of where to
;       find the element to be removed. If this element has
;       no "." in it then the particular element to be removed
;       will be at the highest level.
;
; Example:
;
;IDL> str = {a:1, b:{c:2, d:{e:3}},g:5}
;IDL> nstr = ReplaceElm(str, 'f', 4, SubStructure='b.d')
;IDL> print, nstr
;{   1{   2{   3   4}}   5}
;

```

```

FUNCTION ReplaceElm, str, elm, val, SubStructure=base

```

```

; Catch any errors.
Catch, error_status
IF error_status NE 0 THEN BEGIN
    print, "Couldn't replace element of structure."

```

```
Return, str
ENDIF
```

```
; Get the names of the elements in str
names = StrLowerCase(Tag_Names(str))
element = StrLowerCase(elm)
```

```
IF N_Elements(base) NE 0 THEN BEGIN
  pos = StrLowerCase(StrTrim(Str_Sep(base, '.'), 2))
```

```
  FOR i=0, N_Tags(str)-1 DO BEGIN
    type = Size(str.(i))
    IF type[N_Elements(type)-2] EQ 8 AND names[i] EQ pos[0] THEN
BEGIN
    ; Recurse into the structure.
    IF N_Elements(pos) GT 1 THEN BEGIN
      nb = String(Format='(100(A,;,"."))',pos[1:*])
      substr = ReplaceElm(str.(i), elm, val, SubStructure=nb)
    ENDIF ELSE $
      substr = ReplaceElm(str.(i), elm, val)

    ; Add the substructure to the structure.
    IF N_Elements(newstr) EQ 0 THEN $
      newstr = Create_Struct(names[i], substr) $
    ELSE $
      newstr = Create_Struct(newstr, names[i], substr)
    ENDIF ELSE BEGIN
      ; Add the substructure to the structure.
      IF N_Elements(newstr) EQ 0 THEN $
        newstr = Create_Struct(names[i], str.(i)) $
      ELSE $
        newstr = Create_Struct(newstr, names[i], str.(i))
      ENDELSE
    ENDFOR
  ENDIF ELSE BEGIN
```

```
    FOR i=0, N_Tags(str)-1 DO BEGIN
      IF names[i] NE element THEN BEGIN
        IF N_Elements(newstr) EQ 0 THEN $
          newstr = Create_Struct(names[i], str.(i)) $
        ELSE $
          newstr = Create_Struct(newstr, names[i], str.(i))
        ENDIF
      ENDFOR
```

```
    ; Add the new element
    newstr = Create_Struct(newstr, element, val)
  ENDELSE
```

```
Return, newstr
END
```

```
;
;
; This function will take a structure and return a string array where
; each element represents a particular element in the structure. It
; will recurse substructures as well.
;
; Example:
; IDL> str = {a:1, b:{c:FltArr(2), d:{e:3, f:4}},g:5}
; IDL> print, StructElmNames('str', str)
; str.a str.b.c str.b.d.e str.b.d.f str.g
;
```

```
FUNCTION StructElmNames, sname, struct
```

```
; Get a list of all of the elements in the structure.
elms = StrLowerCase(Tag_Names(struct))

; Build a string array of elements and their values.
FOR i=0L, N_Tags(struct)-1 DO BEGIN
  ; Figure out what type of element we are dealing with.
  s = Size(struct.(i))
  type = s[N_Elements(s)-2]

  IF type EQ 8 THEN BEGIN
    ; The element was another structure so recurse.
    IF N_Elements(outstr) EQ 0 THEN $
      outstr = [StructElmNames(sname+'.'+elms[i], struct.(i))] $
    ELSE $
      outstr = [outstr,StructElmNames(sname+'.'+elms[i],
struct.(i))]
  ENDIF ELSE BEGIN
    ; The element was something other than a structure.
    IF N_Elements(outstr) EQ 0 THEN $
      outstr=[sname+'.'+elms[i]] $
    ELSE $
      outstr=[outstr,sname+'.'+elms[i]]
  ENDELSE
ENDFOR

Return, outstr
```

END

--

Richard D. Hunt
//_/_/_/_/_/_/_/_ SANDIA NATIONAL LABORATORIES _/_/_/_/
//_/_/_/_/_/_/_/_ P.O. Box 5800 M/S 0965 _/_/_/
//_/_/_/_/_/_/_/_ Albuquerque, NM 87185-0965 _/_/_/_/_/_/_/_/_/
//_/_/_/_/_/_/_/_ Voice: (505)844-3193 _/_/_/_/_/_/_/_/_/
//_/_/_/_/_/_/_/_ Fax: (505)844-5993 _/_/_/_/_/_/_/_/_/_/
E-Mail: rdhunt@sandia.gov _/_/_/_/_/_/_/_/_/_
