Rocket software

Rocket Uniface Library 10.4

Example: Looping Over Structs

This example shows how to loop over a Struct to perform actions on the various levels.

The example calls two other functions:

- `REMOVE_TAGS` implements a simple loop that recursively removes the tags on a Struct and all its members and sub-members.
- `PRINT_STRUCT` implements a slightly more complex loop that prints a representation of the Struct to the message frame that is similar to `$dbgstring`.

```
function STRUCT_LOOPS
variables
  struct vStruct
endvariables

  call printHeader("STRUCT_LOOPS") ; display entry header in the message frame

  ; Create a Struct:
  vStruct->$name = "Demo Struct"
  vStruct->$tags->purpose = "sample"
  vStruct->description = "Sample Struct"
  vStruct->subNode = $newstruct
  vStruct->subNode->$tags->purpose = "sample too"
  vStruct->subNode->a = "AAA"
  vStruct->subNode->b = "BBB"

 ; Print the Struct
  putmess "Print the whole Struct:"
  call PRINT_STRUCT(vStruct , "    ") [1]

  ; Remove the tags from all levels of the Struct
  call REMOVE_TAGS(vStruct)
  putmess "%%^After removing tags from the Struct:"

  ; Pass the Struct's collection of children for printing: [2]
  putmess "%%^Print the collection of children of the Struct:"
  call PRINT_STRUCT(vStruct1->*, "    ")

end ;- function STRUCT_LOOPS
```

1. The newly-created Struct looks like this (as rendered by the PRINT_STRUCT function):

   ```
   Print the whole Struct:
    [Demo Struct]
    [$tags]
    [purpose] = "sample"
    [description] = "Sample Struct"
    [subNode]
    [$tags]
    [purpose] = "sample too"
    [a] = "AAA"
    [b] = "BBB"
   ```

2. After calling REMOVE_TAGS to strip out the annotations, the PRINT_STRUCT function shows that the annotations have been removed:

```
After removing tags from the Struct:
 Print a collection of the children of the Struct:
 [description] = "Sample Struct"
 [subNode]
 [a] = "AAA"
 [b] = "BBB"
```

REMOVE_TAGS

This function loops over a Struct collection, recursively stripping tags from the Struct and all its children.

The Struct passed in is a single Struct with multiple members. The tags are removes from that Struct, but not from its children, so the function calls itself recursively.

```
function REMOVE_TAGS
   params
  struct pStruct: in  ; A reference can refer to one or more Structs
endparams
variables
  numeric I, N
endvariables

  if (pStruct->$collSize > 1) ; Multiple Structs in the collection [1]
    I = 1
    N = pStruct->$collSize
    while (I <= N)
      ; Call this entry recursively for each Struct in the collection
      call REMOVE_TAGS(pStruct{I})
      I = I + 1
    endwhile
  elseif  (pStruct->$collSize = 1); Single Struct, so strip tags:
    pStruct->$tags->*->$parent = ""  [2]

    ; Recursively call this entry for all children of the current Struct.
    ; This is a single call, passing all children to the next level:
    if (pStruct->$membercount > 0) call REMOVE_TAGS(pStruct->*) [3]

 else ; Zero Structs, so do nothing

 endif

end ;- function REMOVE_TAGS
```

1. When a collection is passed in, the function calls itself for each Struct in the collection.
2. When a single Struct is passed in (or, more precisely, one collection with `$collsize = 1`), the tags are stripped from that Struct. The action is applied to individual Structs only.
3. If the Struct has children (`$membercount > 0`), the function calls itself for each member.

PRINT_STRUCT

The structure of this function is similar to `REMOVE_TAGS`, but it prints a representation of the Struct to the message

frame that is similar to `$dbgstring`.

> **Note:** `$dbgstring` is slightly richer in functionality and more efficient than this ProcScript implementation, but this example shows how you can write a custom Struct print routine. Whereas the output of `$dbgstring` may change over Uniface versions, a custom implementation enables you to determine a specific format yourself.

```
function PRINT_STRUCT
params
  struct pStruct: in  ; A reference can refer one or more structs
  string pMargin: in  ; Initial margin + indenting for deeper levels
endparams
variables
  numeric I, N
endvariables

  #comment using a define for double quotes to improve readability:
  #define DQ %%"%%%

  if (pStruct->$collSize > 1) ; Multiple structs in the collection [1]
    ; This routine calls itself for each individual Struct in the collection:
    I = 1
    N = pStruct->$collSize
    while (I <= N)
      ; Call this routine recursively for each Struct in the collection
      call PRINT_STRUCT(pStruct{I}, pMargin)
      I = I + 1
    endwhile
  else  ; Single struct, so start printing: [2]

    ; Assume we want to print the Struct, similar to $dbgstring output:
    ; Start printing
    if (pStruct->$isTags) [3]
      putmess $concat(pMargin, "[$tags]", pStruct)  ; Use label '$tags'
    elseif (pStruct->$isLeaf)
      if (pStruct->$isScalar) [4]
        ; Leaf is already printed on preceding level:
        if (!pStruct->$parent->$isLeaf)
          ; Scalar, so print: value
          putmess $concat(pMargin, "<DQ>%%pStruct%%%<DQ>")
        endif
      else
        ; Leaf, so print: [name] = value
        putmess $concat(  %\
            pMargin, "[", pStruct->$name, "] = <DQ>%%pStruct%%%<DQ>") [5]
      endif
    else
      ; Complex Struct, so print: [name]
      putmess $concat(pMargin, "[", pStruct->$name, "]")
    endif

    ; Print tags, if applicable:
    if (pStruct->$tags->$memberCount > 0)
      call PRINT_STRUCT(pStruct->$tags, $concat(pMargin, "  ")) [6]
    endif
```

```
    ; End printing


    ; Recursively call this entry for all children of the current Struct.
    ; This is a single call, passing all children to the next level:
    if (pStruct->$membercount > 0)
      call PRINT_STRUCT(pStruct->*, $concat(pMargin, "  ")) [7]
    endif
  endif
end ;- function PRINT_STRUCT
```

1. When a collection is passed in, the function calls itself for each Struct in the collection.
2. When a single Struct is passed in (or, more precisely, one collection with `$collsize = 1`), it is printed. Struct functions `$istags` and `$isLeaf` are used to determine the nature of the Struct so that the output can be formatted correctly.
3. The Struct is a tag, so print a `[$tags]` label.
4. If the Struct is a scalar Struct, print it. This is needed only when handling a Struct that has both a scalar value and children. For more information, see Struct Leaves.
5. If the Struct is a leaf, print the name and value.
6. If the Struct has tags, recursively call the function to print them.
7. If the Struct has children (`$membercount > 0`), the function calls itself for each member.

Related concepts

$istags

$isLeaf

$isScalar

$concat