



Rocket Uniface Library 10.4

Example: Struct Basics

This ProcScript example demonstrates how multiple references access the same Struct, and that a Struct is deleted when there is no longer a struct variable that refers to it or any of its parents.

```
function STRUCT_BASICS
variables
    struct vStruct, vPerson ; declare struct variables
endvariables

; -- Build up a Struct --
; Create a Struct
vStruct = $newstruct [1]

; Create Struct members: [2]
vStruct->group = $newstruct
vStruct->group->person = $newstruct
vStruct->group->person->firstname = "John"
vStruct->group->person->email = "john@home.com"

; Display the Struct in the message frame:
call printHeader("STRUCT_BASICS") ; display entry header in the message frame
putmess "Struct referred to by vStruct: "
putmess vStruct->$dbgstring [3]

; -- Update the Struct using a different variable --
vPerson = vStruct->group->person [4]

putmess "Struct referred to by vPerson: "
putmess vPerson->$dbgstring [5]

vPerson->email = "john.smith@home.com" [6]

putmess "Struct referred to by vStruct: "
putmess vStruct->$dbgstring [7]
putmess "Note: 'email' changed with vPerson is also changed in vStruct: "

; -- Delete a Struct --
; Display the parent of vPerson [8]
putmess "The parent of vPerson is: [%(vPerson->$parent->$name)%%]"

; Assign an empty Struct to vStruct, and display the Structs
vStruct = $newstruct [9]
putmess "After assigning a new value to vStruct, vStruct points to empty Struct:"
putmess vStruct->$dbgstring
putmess "vPerson is still valid: [%(vPerson->firstname)%%]"
putmess "but the parent of vPerson is now invalid: [%(vPerson->$parent->$name)%%]"
putmess "resulting in a ProcScript error: %%$procerror: %%$item("DESCRIPTION", %\
    $procerrorcontext)"

end ; function STRUCT_BASICS
```

The code builds up the Struct, updates the Struct using a different **struct** variable, then deletes a Struct.

1. Create a new empty Struct and assigns it to the struct variable. However, this is not strictly required. The Struct is created on the fly when a member is assigned to the variable.
2. Create the members of a Struct. This must be done explicitly; members are not created on the fly.
3. Display a representation of the Struct in the message frame using the `$dbgString` (or `$dbgstringplain`) Struct function.

```
Struct referred to by vStruct:
[]
[group]
[person]
[firstname] = "John"
[email] = "john@home.com"
```


4. Get a reference to the person node and assign it to the **struct** variable vPerson.
5. Display a representation of the person Struct in the message frame:

```
Struct referred to by vPerson:
[person]
[firstname] = "John"
[email] = "john@home.com"
```

6. Change the value of the email member of the person node referred to by vPerson.
7. Display a representation of the main Struct (vStruct) in the message frame:

```
Member 'email' changed with vPerson is also changed in vStruct:
[]
[group]
[person]
[firstname] = "John"
[email] = "john.smith@home.com"
```

8. Show that the parent Struct of the person member can be accessed using vPerson.

 **Note:** When using the `->` operator in string substitutions, the expression must be enclosed within parentheses.

```
The parent of vPerson is: [group]
```

9. Assign a new member to the vStruct variable. This deletes the Struct that it referred to before. The person Struct remains but has lost its outer context (its direct parent and above), so an attempt to get its parent results in an error. After assigning a new value to vStruct:

vStruct points to empty Struct: `[] = ""`

but the parent of vPerson is now invalid: `[]`

resulting in a ProcScript error: `-84: Not a valid Handle or Struct specified`

Related concepts

[Creating and Deleting Structs](#)

[Assigning Values to Structs](#)

[\\$dbgString](#)

[\\$dbgStringPlain](#)

[\\$newstruct](#)

[\\$parent](#)