Rocket software

Rocket Uniface Library 10.4

Example: Struct Conversions

This example uses Struct manipulation to transform incoming XML data into Uniface entities so that the data can be updated, and transform it back to original XML format.

To run this example, you need to paint the following entities in your test component:

- Outer entity: BOOKS.MODELX, with the fields: TITLE, CATEGORY, DESCRIPTION
- Inner entity: AUTHOR.MODELX, with the field : NAME

> **Note:** For your convenience, you can copy all the code in this example, and in those included under Working With Structs: Code Examples, from Struct Code Examples, and paste it into a Uniface trigger (such as the EXEC trigger of a test form). Compile the form and test it to see the results.

The incoming XML data is defined as a `samplexml:blockdata`, which can be referenced by `$samplexml`:

```
samplexml:blockdata @
<?xml version="1.0"?>
<catalog>
 <book id="ref2451">
 <author>Smith, John</author>
 <title>My Road</title>
 <genre>Biography</genre>
 <publisher>ABC books</publisher>
 <publication_date>2010-07-21</publication_date>
 <description>Autobiography by John Smith</description>
 <price>22.90</price>
 </book>
 <book id="ref7836">
 <author>Black, E.J.</author>
 <title>Summer recipes</title>
 <genre>Cookery</genre>
 <publisher>Black Bee Publishers</publisher>
 <publication_date>2010-08-12</publication_date>
 <description>Recipes by top chef E.J. Black</description>
 <price>15.00</price>
 </book>
</catalog>
@
```

```
entry STRUCT_CONVERSIONS
variables
  struct vStruct, vOutStruct
  string vOutXML
  numeric I
  string vWarning
endvariables

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

  putmess "The starting point is the following XML sample:%%^%%%"
  ; $samplexml is defined as blockdata at the end of the entry
```

```
  putmess $samplexml
; ====================================================
; Convert XML to Struct
; ====================================================
  xmlToStruct vStruct, $samplexml [1]

  putmess "Struct, as converted from XML (without tags):"
  putmess vStruct->$dbgstringplain [2]
  ; Note: In this example, the tags will not be used when converting
  ;       back to XML so they could be deleted. [3]

; ====================================================
; Transform the Struct to match the required Uniface structure: [4]
; ====================================================
  ; 1. Change the structure for the outer entity
     ; FROM: vStruct->catalog->book TO: vStruct->catalog->BOOKS->OCC

     ; Add one more level BOOKS
  vStruct->catalog->BOOKS = $newstruct

     ; Put all 'book' Structs into it
  vStruct->catalog->book->$parent = vStruct->catalog->BOOKS

     ; Rename the 'book Structs to "OCC"
  vStruct->catalog->BOOKS->book->$name = "OCC"

  ; 2. Rename fields to match those in BOOKS, where necessary
  vStruct->catalog->BOOKS->OCC->genre->$name = "CATEGORY"

  ; 3. Change the structure for the inner entity
     ; Add a new level for the inner entity AUTHOR
  vStruct->catalog->BOOKS->OCC->AUTHOR = $newstruct
  vStruct->catalog->BOOKS->OCC->AUTHOR->OCC = $newstruct

     ; For each BOOKS occurrence, move the author Struct to this new inner Struct:
  I = 1
  while (I <= vStruct->catalog->BOOKS->OCC->$collsize)
    vStruct->catalog->BOOKS->OCC{I}->author->$parent =        %\
                  vStruct->catalog->BOOKS->OCC{I}->AUTHOR->OCC
    I = I + 1
  endwhile

     ; Rename the XML fields to match those in the entity AUTHOR

  vStruct->catalog->BOOKS->OCC->AUTHOR->OCC->author->$name = "NAME"

  putmess "Struct modified to match component structure:"
  putmess vStruct->$dbgstringplain [5]

; ====================================================
; Convert the Struct to the Uniface component structure
; ====================================================
  ; This is only possible if the component performing the transformation
  ; contains a component structure that matches the Struct, because the
  ; data is inserted into the component.
```

```
  structToComponent vStruct->catalog->BOOKS [6]

  ; Warnings are generated for entities and fields not found in the component:
  if ($procerror < 0) [7]
    putmess "Failed to execute 'structToComponent': ProcScript error:%%$procerror%%%"
    putmess $procerrorcontext
  else
    ;Warnings are generated for unrecognized fields:
    if ($item("WARNINGS", $procreturncontext) > 0)
      putmess "Warnings after execution of 'structToComponent':"
      I = 1
      while (I <= $item("WARNINGS", $procreturncontext))
        vWarning = $itemnr(I, $item("DETAILS", $procreturncontext))
        putmess "  Warning %%i%%%:"
        while (vWarning != "")
          putmess "    %%$itemnr(1, vWarning)%%%"
          delitem vWarning, 1
        endwhile
        I = I + 1
      endwhile
    endif
   endif

  ; Regular Uniface processing can now may take place. [8]

; ======================================================
; Convert the component data to a Struct
; ======================================================
    componentToStruct vOutStruct [9]

  ; Once again, to restrict the output in the message frame,
  ; you can use $dbgstringplain or remove the tags:
  ; call REMOVE_TAGS(vOutStruct)

  putmess "%%^%%%"
  putmess "Struct after conversion from component:"
  putmess "(This requires that the required entities/fields are present in the component)"
  putmess vOutStruct->$dbgstringplain [10]

; ======================================================
; Reconstruct the original XML format: [11]
; ======================================================

  ; Move the author details from the AUTHOR level up to the BOOK level:
  ; 1. Rename the 'name' field (could also be done after the move)
  vOutStruct->BOOKS.MODELX->OCC->AUTHOR.MODELXNAME->OCC->NAME->$name = "author"

  ; 2. Move each occurrence up one level:
  I = 1
  while (I <= vOutStruct->BOOKS.USTRUCTS->OCC->$collsize)
    vOutStruct->BOOKS.MODELX->OCC{I}->AUTHOR.MODELX->OCC->author->$parent =  %\
                                        vOutStruct->BOOKS.MODELX->OCC{I}
    I = I + 1
  endwhile

  ; 3. Detach AUTHOR from its parent:
```

```
  vOutStruct->BOOKS.USTRUCTS->OCC->AUTHOR.USTRUCTS->$parent = "" ;- coll. upd.

  ; 4. Rename the Struct members:
  vOutStruct->BOOKS.USTRUCTS->$name = "catalog" ;- use BOOKS for the "catalog"
  vOutStruct->catalog->OCC->$name = "book" ;- collection update
  vOutStruct->catalog->book->TITLE->$name = "title" ;- just the case
  vOutStruct->catalog->book->CATEGORY->$name = "genre" ;- collection update

  ; alternative place to rename the AUTHORS' "NAME" field:
  ; vOutStruct->catalog->book->NAME->$name = "author" ;- collection update
  vOutStruct->catalog->book->DESCRIPTION->$name = "description" ;- the case

  ; 5 Add the XML declaration:
  vOutStruct->$tags->xmlVersion="1.0"

  ; 6. Make the top-level struct nameless, so that it
  ;    will not be included as an additional level in the XML:
  vOutStruct->$name = ""

  ; Convert the Struct to XML:
  structToXml vOutXML, vOutStruct [12]
  putmess "The XML after conversion back from the component"
  putmess vOutXML [13]

  ; Assuming the component is a form, to see the data use the edit statement
  ;edit

end ;- entry STRUCT_CONVERSIONS
```

1. Convert the XML data to a Struct using the `xmlToStruct` ProcScript statement. For more information, see xmlToStruct.
2. The XML tags can be relevant when converting back to XML, but they are distracting when debugging the Struct. To display the Struct without tags, use `$dbgstringplain`.

```
Struct, as converted from XML (without tags):
[]
 [catalog]
 [book]
 [id] = "ref2451"
 [author] = "Smith, John"
 [title] = "My Road"
 [genre] = "Biography"
 [publisher] = "ABC books"
 [publication_date] = "2010-07-21"
 [description] = "Autobiography by John Smith"
 [price] = "22.90"
 [book]
 [id] = "ref7836"
 [author] = "Black, E.J."
 [title] = "Summer recipes"
 [genre] = "Cookery"
 [publisher] = "Black Bee Publishers"
 [publication_date] = "2010-08-12"
 [description] = "Recipes by top chef E.J. Black"
 [price] = "15.00"
```

3. In this example, the tags are being ignored, so you can choose to remove them from the Struct. For example, you can, remove the `$tags` Structs by changing their parents using `$parent`. The following instructions update the collection of Structs are multiple levels:

```
vStruct->$tags->*->$parent = "" ; Updated collection
 vStruct->*->$tags->*->$parent = "" ; Update collecion in collection...
 vStruct->*->*->$tags->*->$parent = "" ; etc.
 vStruct->*->*->*->$tags->*->$parent = ""
```

   Or you could call the `REMOVE_TAGS()`function. For more information, see Example: Looping Over Structs.

```
call REMOVE_TAGS(vStruct)
```

4. When transforming the Struct from the XML structure to the Uniface structure, you need to decide what to do with members that are not required in the data structure of the component, such as publisher and price. You can choose to remove the members that correspond to these elements, but in this example, they are left in the Struct. This will generate warnings in `$procreturncontext`.
5. The Struct after it has been modified to match the component structure:

```
Struct modifed to match component structure:
[]
 [catalog]
 [BOOKS]
 [OCC]
 [id] = "ref2451"
 [title] = "My Road"
 [CATEGORY] = "Biography"
 [publisher] = "ABC books"
 [publication_date] = "2010-07-21"
 [description] = "Autobiography by John Smith"
 [price] = "22.90"
 [AUTHOR]
 [OCC]
 [NAME] = "Smith, John"
 [OCC]
 [id] = "ref7836"
 [title] = "Summer recipes"
 [CATEGORY] = "Cookery"
 [publisher] = "Black Bee Publishers"
 [publication_date] = "2010-08-12"
 [description] = "Recipes by top chef E.J. Black"
 [price] = "15.00"
 [AUTHOR]
 [OCC]
 [NAME] = "Black, E.J."
```

6. Convert the Struct to the Uniface component structure using the `structToComponent` ProcScript statement.
7. Because the Struct still contains members that do not match the component structure, warnings are issued and details provided in `$procerrorcontext`. This code handles these error, displaying the in the message frame.

```
Warnings after execution of 'structToComponent':
 Warning 1:
 SEVERITY=Warning
```

```
 ID=-1161
 MNEM=<USTRUCTERR_NO_MATCHING_NAME>
 DESCRIPTION=No matching name found during conversion from struct
 CURRENTSTRUCT=BOOKS->OCC{1}->id{1}
 ADDITIONAL=SPECIFIEDNAME=id;EXPECTEDTYPE=entity or field
 Warning 2:
 SEVERITY=Warning
 ID=-1161
 MNEM=<USTRUCTERR_NO_MATCHING_NAME>
 DESCRIPTION=No matching name found during conversion from struct
 CURRENTSTRUCT=BOOKS->OCC{1}->publisher{1}
 ADDITIONAL=SPECIFIEDNAME=publisher;EXPECTEDTYPE=entity or field
... etc.
```

8. Once the data has been inserted into the component, normal Uniface processing can take place. This can include modifying (in ProcScript or by ta user) and storing the data.
9. When processing is complete, the data can be converted back to a Struct using the `componentToStruct` ProcScript statement.
10. Struct after conversion from a Uniface component:

```
Struct after conversion from component:
(This requires that the required entities/fields are present in the component)
[USTRUCTS_FRM]
 [BOOKS.MODELX]
 [OCC]
 [TITLE] = "My Road"
 [CATEGORY] = "Biography"
 [DESCRIPTION] = "Autobiography by John Smith"
 [AUTHOR.MODELX]
 [OCC]
 [NAME] = "Smith, John"
 [OCC]
 [TITLE] = "Summer recipes"
 [CATEGORY] = "Cookery"
 [DESCRIPTION] = "Recipes by top chef E.J. Black"
 [AUTHOR.MODELX]
 [OCC]
 [NAME] = "Black, E.J."
```

11. The Struct then needs to be transformed back to a format that matches the XML, reversing the previous steps.
12. The Struct can be converted to XML using the `structToXML` ProcScript statement.
13. The XML produced looks like this:

```
The XML after conversion back from the component
<?xml version="1.0"?>
<catalog>
 <book>
 <title>My Road</title>
 <genre>Biography</genre>
 <description>Autobiography by John Smith</description>
 <author>Smith, John</author>
 </book>
 <book>
```

```
 <title>Summer recipes</title>
 <genre>Cookery</genre>
 <description>Recipes by top chef E.J. Black</description>
 <author>Black, E.J.</author>
 </book>
</catalog>
```

Note: Some XML elements, such as `<publisher>`, are now lost, but this was a deliberate choice.

Related concepts

- Transforming Complex Data Using Structs
- Working With Structs: Code Examples
- xmlToStruct
- structToXml
- structToComponent
- componentToStruct