Rocket Uniface Library 10.4

Struct Leaves

A Struct leaf is a Struct member that has no value, or only a single scalar value. However, adding another value (member) to a Struct leaf implicitly changes it to a nested Struct. The value of the former leaf is now held in a nameless Struct, known as a *scalar Struct*, and the nested Struct has a mixed data structure that may require special handling.

Value of a Struct Leaf

The value of a Struct leaf is returned when you refer to the Struct member. For example, the following statement puts the value of the member `myLeaf` in the message frame:

```
putmess struct1->myLeaf
```

In fact, the value is treated as a member of the leaf, so the function `$membercount` typically returns `1` for a leaf.

However, nested Structs can also have a single member, and sometimes it is important to distinguish between Struct leaves and nested Structs. In such cases, you can use the Struct function `$isLeaf`, which returns `1` for a leaf, and `0` for a non-leaf.

In most cases, this is all you need to know to understand and work with Struct leaves. However, if the Struct embodies a mixed data content model, which combines scalar values with complex values, you need to be able to distinguish between the scalar and non-scalar members. Mixed content is not possible in Uniface component structures, but it is supported by XML and common in XHTML.

Scalar Structs

A scalar Struct is used only to contain the scalar value. It cannot have child nodes, and it is not possible to use access operators (such as `->`) on a scalar Struct. To distinguish a scalar Struct from a Struct node, you can use the `$isScalar` Struct function, which returns `1` for scalar Structs and `0` for Struct nodes.

Important: Most ProcScript does not deal with scalar Structs. For example, a process that loops through the children of a Struct, drilling down to the children that are Structs themselves, should stop where it encounters a Struct leaf. During such loops, test for `$isLeaf` and do not request the members of a leaf. Also `$dbgstring` and `$dbgstringplain` do not display the leaf value (a scalar Struct) as an actual Struct, but merely as a value.

When a Struct leaf is assigned an additional Struct member and becomes a nested Struct, the value remains the same. More precisely, the value of a Struct is the concatenated value of all of its direct members that are scalar Structs. Thus, for the following XHTML code, the scalar value would be `Text can be or`:

```
<div>Text can be <b>bold</b> or <em>italic</em></div>
```

In the following representation, the nameless Struct members are scalar Structs:

```
[]
 [div]
 "Text can be "
 [b] = "bold"
 "or "
 [em] = "italic"
```

You can retrieve all the scalar members of a Struct using the Struct function `$scalar`.

To illustrate the dynamic nature of Struct leaves, consider how to build the following piece of HTML. (part of a table) using Structs.

```
<td>John <b>Smith</b></td>
```

The element `td` has mixed content, containing both text (`John `), and an element (`b`) with only text (`Smith`).

1. First, create a Struct member `td` with value `John` :

   ```
   mystruct->td = "John "
   ```

   At this point, `td`is a leaf, so `mystruct->td->$isLeaf`is `1`

2. Verify this by displaying the Struct in the message frame:

   ```
   putmess mystruct->$dbgstring
   ```

   Because `td`is a leaf, `John`is displayed as a value:

   ```
   []
    [td] = "John "
   ```

3. Now add a member to represent the bold element containing `Smith`:

   ```
   mystruct->td->*{-1} = $newstruct ; create new member under td
   mystruct->td->*{-1}->$name = "b" ; assign the name
   mystruct->td->b = "Smith" ; assign the value
   putmess mystruct->$dbgstring
   ```

   `td`is now a complex node, so `mystruct->td->$isLeaf`is `0`

   The message frame shows `"John "`as a member, rather than as a mere value:

   ```
   []
    [td]
    "John "
    [b] = "Smith"
   ```

Tip: A more straight-forward way to build this HTML construct is with the following code:

```
mystruct->td = "John " 1
mystruct->td->b = "Smith" 2
```

1 Create a member `td` with value `John`. At this point, `td` is a leaf, so `mystruct->td->$isLeaf` is `1`

2 Add a member `b` to `td`, with value `Smith`. Now `td` is a complex node, so `mystruct->td->$isLeaf` is `0`

Mixed content Structs can also be created by moving one Struct to another. For example:

```
mystruct->td = "John " 1
tmp->b = "Smith" 2
tmp->b->$parent = mystruct->td 3
```

1 Create a member `td` with value `John`. At this point, `td` is a leaf, so `mystruct->td->$isLeaf` is `1`

2 Create another struct with member band value `Smith`. The Struct `tmp` is a placeholder on which a new member named `b` with value `Smith` is created

3 Use `$parent` to move the member `b` to `mystruct`. By default, the member is appended after the last member, so there is no need to set `$index` to `2`.

Effect of XML Attributes on Struct Leaves

Other XML constructs can result in a Struct member being treated as a Struct node instead of a Struct leaf. For example, XML elements can have attributes, which are handled as child elements of their parent element. For example:

```
<div class="note">Text can be bold</div>
```

The `$dbgstringplain` representation looks like this:

```
[]
 [div]
 [class] = note
 Text can be bold
```

The following table shows the values returned by Struct functions for the `div` member, indicating that `div` is a node with 2 members:

Struct Function	Returned Value
`vStruct->div->$isLeaf`	`0`
`vStruct->div->$isScalar`	`0`
`vStruct->div->$memberCount`	`2`

The following table shows the values returned by Struct functions for the nameless scalar member. It shows that it is a scalar Struct with no members:

Struct Function	Returned Value
`vStruct->div->*{2}->$isLeaf`	`1`

Struct Function	Returned Value
`vStruct->div->*{2}->$isScalar`	`1`
`vStruct->div->*{2}->$memberCount`	`0`

Related concepts

- $isLeaf
- $isScalar
- $scalar
- $memberCount