Rocket Uniface Library 10.4

structToXml

Convert a Struct to XML, without a schema.

`structToXml` *XmlTarget*, *StructSource*

Example: `structToXml vOutputXml, vStruct`

Parameters

Table: Parameters

Parameter	Data Type	Description
XmlTarget	`xmlstream`	Variable or parameter to which the converted XML is written; must be of type `xmlstream`, `string`, or `any`.
StructSource	`struct`	Variable or parameter referring to the source Struct; must be of type `struct` or `any`.

Return Values

Table: Values Returned in `$procerror` after `structToXml`

Error	Meaning
`0`	XML document successfully created. However, the XML produced may not be what is expected if non-fatal errors occurred during conversion, because everything that is not recognized or usable is ignored. Warnings about such conditions are made available in `$procReturnContext`. See $procReturnContext for structToXml.
`<0`	An error occurred. `$procerror` contains the exact error.

Table: Values Commonly Returned by `$procerror`

Error Number	Error Constant	Meaning
`-1905`	`STRUCTERR_INPUT`	Input struct data is not valid. For example, the `struct` variable may have been declared, but not initialized.

Table: Errors and warnings returned in `$procreturncontext`

Error	Error Constant	Meaning
`-1160`	`USTRUCTERR_TAGVALUE_NOT_APPLICABLE`	Annotation `xmlClass` has no value, or an unknown or illegal value (based on the current context)

Use

Allowed in all component types.

Description

XML annotations drive the way in which `structToXml` performs the conversion. Should a Struct be converted to an element, an attribute, or a declaration? Do namespace values need to be added? Without the value of the `xmlClass` and other annotations, there is no way for `structToXml` to determine this. When preparing a Struct for conversion to XML, you therefore need to ensure that you set the `xmlClass` annotation for each member of the Struct.

The `structToXml` conversion routine processes the Struct members in the order in which they appear. You therefore need to ensure that the Struct members are in the correct sequence. Thus, if the source Struct has `xmlClass` set to `element`, its members must be ordered so that members with `xmlClass=attribute` or `xmlClass=namespace-declaration` come before `xmlClass=comment`.

XML Documents and Snippets

`structToXml` can generate either an XML document (that is, well-formed XML with a single root node) or partial XML documents (well-formed XML with multiple root nodes), known as snippets.

`structToXml` handles *StructSource* as an XML document if the `xmlClass` of the source Struct is set to `document`, or if the source Struct is nameless and contains only one member. For example, the following Struct has a single nameless source node containing a single named member (`movie`):

```
[]
 [movie] = "The Matrix"
 $tags
 xmlClass = "element"
 [p] = "The Matrix is a great movie."
 $tags
 xmlClass = "element"
 [p] = "It is much better than its sequels."
 $tags
 xmlClass = "element"
```

This is treated as if the source struct was tagged as `xmlClass = "document"`

```
[]
 $tags
 xmlClass = "document"
 [movie] = "The Matrix"
 ...
```

`structToXml` handles *StructSource* as snippets if the source struct contains a collection of Structs. For example, the following shows a collection of Structs produced by `vStruct->p`:

```
[]
[p] = "The Matrix is a great movie."
 $tags
 xmlClass = "element"
[p] = "It is much better than its sequels."
 $tags
 xmlClass = "element"
```

Converting Struct to XML

When preparing a Struct for conversion to XML, keep the following in mind:

- Each Struct member must have an `xmlClass` that defines the XML construct to which it should be converted. For more information, see Struct Annotations for XML.

  The `xmlClass` is optional in the following cases:

  - The source (top-level) Struct is nameless and contains a single member; `xmlClass` is assumed to be `document`.
  - Non-scalar Struct members; `xmlClass` is assumed to be `element`.
  - Struct members named `#comment` ; `xmlClass` is assumed to be `comment`.
- Depending on the value of `xmlClass`, `structToXml` will read and convert other XML annotations that are valid for the XML construct.
- If `xmlClass` holds an unknown value, the Struct member is ignored and a warning (`STRUCTERR_TAGVALUE_NOT_APPLICABLE`) is returned in `$procReturnContext`.
- If `xmlClass` holds an illegal value, based on the Struct's context, a warning is returned in `$procReturnContext`. The context is determined by the position of the Struct in relation to a valid XML document.
- If a Struct represents an XML snippet (it holds a collection of Structs), the `xmlClass` of each member can be set to `comment`, `processing-instruction`, or `element`.
- If a Struct represents an XML document:
  - The Struct may optionally have `xmlClass` set to `document`.
  - The Struct may optionally have annotations `xmlVersion`, `xmlEncoding`, and `xmlStandAlone`, which are used to generate an XML declaration. For more information, see XML Declaration.
  - The Struct can have no, one, or multiple members representing comments, processing instructions, or elements(`xmlClass` set to `comment`, `processing-instruction`, or `element`).
  - The Struct may optionally have one member representing a DOCTYPE declaration (`xmlClass` set to `doctype`).
- If a Struct represents a DOCTYPE declaration (`xmlClass` set to `doctype`):
  - The Struct can only be a child of an XML document struct.
  - The Struct must occur before Structs with `xmlClass` set to `element`.
  - The Struct may contain other Structs for element and entity declarations, attribute lists, and notation declarations. For more information, see XML DOCTYPE Declaration.
  - If there are multiple `doctype` Structs, or they are in the wrong position within the XML document, a warning is returned.
- If a Struct represents an element (`xmlClass` set to `element`). it can have no, one, or multiple members for attributes (`xmlClass` set to `attribute` or `namespace-declaration`). If these are present, they must occur before any of the following members:
  - Scalar member for character data (no `xmlClass` annotation)
  - Scalar member for CDATA (`xmlClass` set to `CDATA`)
  - Struct node for a child element (`xmlClass` set to `element`)
  - Struct member for a comment (`xmlClass` set to `comment`, and optionally named `#comment`)
  - Struct member for processing instruction (`xmlClass` set to `processing-instruction`)
- If the data type of the Struct member is `string`, it is put into the XML document as is, without any reformatting.

  If the data type is something other than string, the data is formatted according to the schema data type specified in the `xmlDataType` and `xmlTypeNamespace` tags. If these tags are not present, an appropriate default schema data type is used to format the data in the XML document. For example, if the data type is Raw and `xmlDataType="base64Binary"`, the data is base64 encoded in the resulting XML document. If the data type is Raw but there is no `xmlDataType` tag, the default schema data type is hexBinary, so the data will be put into the XML document in hexadecimal format.

For the `time` and `datetime` data types, the time is always local time without time zone information.

Important: If you specify an `xmlDataType` tag, ensure that its value is appropriate for the Uniface data type of the scalar member, otherwise unpredictable results may occur.

- The order of the annotations is not relevant. However, if a Struct contains multiple annotations with the same name, only the first one is used.

$procReturnContext for structToXml

`$procReturnContext` contains context and error information about the conversion in the form of a nested Uniface list.

Context=structToXml ;}
{Infos=Number ;
{Warnings=Number ;}
{Errors=Number ;}
{DETAILS=ID=MsgNum !!;SEVERITY=Type !!;MNEM=Mnemonic !!;DESCRIPTION=ErrorDescription !!;CURRENTSTRUCT=Struct !!;ADDITIONAL=TAGNAME=Name !!!;TAGVALUE=Value !!!;EXPECTED=ExpectedValue} { !;ID= ...}

Table: Items Returned by `$ProcReturnContext` for `structToXml`

Item	Description
`Context`	Value indicating the previously executed command that set `$procReturnContext`, in this case, `structToXml`.

`Infos`
`Warnings`
`Errors`
	Number of messages, warnings, and non-fatal errors generated during processing.

`DETAILS`	Details about any messages, warnings, and non-fatal errors encountered during processing, structured as a Uniface sublist.
`ID`	Message number.
`MESSAGE`	Message text.
`SEVERITY`	Importance of the issue; one of `INFO`, `WARNING`, or `ERROR`.
`MNEM`	Mnemonic for the specified (numeric) ID.
`DESCRIPTION`	Short description of the issue.
`CURRENTSTRUCT`	List of all preceding parents, starting from the top. Each parent is described by its name (which can be empty) and index number. The top-level parent has no index number.

Item	Description
`ADDITIONAL`	Uniface sublist of additional information about the Struct (member) causing the message. This information is provided if there is more detailed information to report, such as unexpected tags or tag values.
`TAGNAME`	Name of the annotation tag; optional.
`TAGVALUE`	Value of the tag specified by `TAGNAME`.
`EXPECTED`	Expected object for the context; one of: • `Struct valid on XML document level` • `Struct valid on XML element level` • `DTD declaration` • `DTD attribute declaration`

Example: `$ProcReturnContext` after `structToXml`

The following shows the type of information returned in `$procReturnContext` for `structToComponent` (formatted for readablity):

```
Context=StructToXml;
Warnings=1;
DETAILS=
 ID=-1160!!;
 SEVERITY=Warning!!;
 MNEM=<USTRUCTERR_TAGVALUE_NOT_APPLICABLE>!!;
 DESCRIPTION=Struct tag value not applicable in conversion from struct!!;
 CURRENTSTRUCT=""->#comment{1}!!;
 ADDITIONAL=
 TAGNAME=xmlClass!!!;
 TAGVALUE=commen!!!;
 EXPECTED=Struct valid on XML document level
```

Example: Building a Struct and Converting it to XML

Assume that you want to create the following XML message:

```
<!-- message to Jim -->
<message priority="High" from="Reception" to="Jim">Please contact your brother</message>
```

The following ProcScript attempts to build this Struct and convert it to XML, but contains three errors:

```
variables
 struct vStruct
 xmlStream vOutputXml
 string vReturnContext
endvariables
; Build Struct
vStruct = $newstruct
vStruct->"#comment" = "message to Jim"
```

```
vStruct->*{-1}->$tags->xmlClass="comment" 1
vStruct->message = $newstruct
vStruct->message->$tags->xmlClass = "element"
vStruct->message->priority = "High"
vStruct->message->priority->$tags->xmlClass = "attribute"
vStruct->message->*{-1} = "Please contact your brother"
vStruct->message->from = "Reception"
vStruct->message->from->$tags->xmlClass = "attribute" 2
vStruct->to = "Jim"
vStruct->to->$tags->xmlClass = "attribute" 3
; Convert to XML
structToXml vOutputXml, vStruct
; Write the XML to a file
filedump vOutputXml, "generatedXmlDoc.xml"
```

1. Spelling error when tagging the `comment`.
2. Attribute `from` is after the content of `message`, so it is not in the correct order for the XML.
3. Attribute `to` is created as a sibling Struct to message, which puts it on the document level.

The resulting XML looks like this:

```
<message priority="High">Please contact your brother</message>
```

`$procReturnContext` provides information about these errors:

```
Context=StructToXml;
Warnings=3;
DETAILS=
 ID=-1160!!;
 SEVERITY=Warning!!;
 MNEM=<STRUCTERR_TAGVALUE_NOT_APPLICABLE>!!;
 DESCRIPTION=Struct tag value not applicable in conversion from struct!!;
 CURRENTSTRUCT=""->#comment{1}!!;
 ADDITIONAL=
 TAGNAME=xmlClass!!!;
 TAGVALUE=commen!!!;
 EXPECTED=Struct valid on XML document level!;
 ID=-1160!!;
 SEVERITY=Warning!!;
 MNEM=<STRUCTERR_TAGVALUE_NOT_APPLICABLE>!!;
 DESCRIPTION=Struct tag value not applicable in conversion from struct!!;
 CURRENTSTRUCT=""->message{1}->from{1}!!;
 ADDITIONAL=
 TAGNAME=xmlClass!!!;
 TAGVALUE=attribute!!!;
 EXPECTED=Struct valid on XML element level!;
 ID=-1160!!;
 SEVERITY=Warning!!;
 MNEM=<STRUCTERR_TAGVALUE_NOT_APPLICABLE>!!;
 DESCRIPTION=Struct tag value not applicable in conversion from struct!!;
 CURRENTSTRUCT=""->to{1}!!;
 ADDITIONAL=
 TAGNAME=xmlClass!!!;
 TAGVALUE=attribute!!!;
```

```
EXPECTED=Struct valid on XML document level
```

The following example shows the correct ProcScript code:

```
function buildStruct
variables
 struct vStruct
 xmlStream vOutputXml
endvariables
; Build Struct
vStruct = $newstruct
vStruct->"#comment" = "message to Jim"
vStruct->*{-1}->$tags->xmlClass="comment"
vStruct->message = $newstruct
vStruct->message->$tags->xmlClass = "element"
vStruct->message->priority = "High"
vStruct->message->priority->$tags->xmlClass = "attribute"
vStruct->message->from = "Reception"
vStruct->message->from->$tags->xmlClass = "attribute"
vStruct->message->to = "Jim"
vStruct->message->to->$tags->xmlClass = "attribute"
vStruct->message->*{-1} = "Please contact your brother"

; Convert to XML
structToXml vOutputXml, vStruct

; Write the XML to a file
filedump vOutputXml, "generatedXmlDoc.xml"
```

Replacing Escape Sequences with XML Entities

By default, `structToXml` produces escape sequences for GOLD and text formatting characters (bold, italic, underline, and their combinations).

- For GOLD characters: `&#x8FFFF;`. For example, GOLD ; results in `&#x8FFFF;1B`.
- For bold, underline, and italic (and their combinations): `&#x1FFFF; &#x2FFFF; ... &#x7FFFF;`. For example, the word Hi in bold results in `&#4FFFF;H&#4FFFF;i`.

The Uniface DTD includes XML entity definitions for these characters. If you want `structToXml` to produce the entities defined in Uniface DTD instead, you need to ensure that the Struct has a doctype member with the tag `xmlSystemID` that has the value `UNIFACE.DTD`.

Note: This is useful only if you are using `structToXml` to produce an XML file that Uniface can import into the repository. In this case, you must also ensure that the rest of the Struct conforms to the Uniface DTD. For more information, see XML Streams.

For example:

```
mystruct->rootelement{mystruct->rootelement->$index + 1} = $newstruct
mystruct->rootelement{-1}->$index = 1
mystruct->rootelement{1}->$tags->xmlClass = "doctype"
mystruct->rootelement{1}->$tags->xmlSystemID = "UNIFACE.DTD"
```

The resulting XML string includes:

- A `<!DOCTYPE ...>` line specifying the DTD.
- XML entities for special characters such as GOLD characters and text formatting (bold, Italic, underline and their combinations). These XML entities that are defined in the Uniface DTD.

To read an XML file produced this way, use `xmlToStruct` with the `/validate` switch. This ensures that the DTD is read and the XML entities are validated against the DTD.

Related concepts

- Uniface XML Constructs
- Struct Annotations for XML
- Transforming Complex Data Using Structs
- structToXml/schema
- xmlToStruct
- $procReturnContext
- $tags