Rocket Uniface Library 10.4

xmlToStruct

Convert an XML document to a Struct.

```
xmlToStruct {/full} {/whitespace}  StructTarget, XmlDocument

xmlToStruct {/full} {/validate } { /schema}   StructTarget, XmlDocument, SchemaList
```

Example: `xmlToStruct/full vStruct, INPUT_FLD`

Qualifiers

Table: Qualifiers

Qualifier	Description
`/full`	Include comments, doctype declarations, and namespace declarations, if they are present, and leave leading and trailing whitespace in character sequences within elements. By default, these constructs are ignored and whitespace stripped. See Conversion with /full and Namespace Handling.
`/whitespace`	Convert whitespace between elements to Struct members and leave leading and trailing whitespace in element values or comments. By default, such whitespace is ignored or stripped. This qualifier is only applicable if `/validate` or `/schema` are not specified. DTDs and schemas handle whitespace correctly, causing only whitespace that the DTD or schema does not deem significant to be discarded.
`/validate`	Validate the XML source against its DTD or schema. A DTD can be provided as an embedded DTD or as a reference. A schema can be provided as a reference or in the *SchemaList* parameter. A referenced DTD or schema is expected to be located at the provided URL. If only a DTD or schema name is provided, a file with that name is expected in the working directory. By default, the *XmlSource* is only checked to ensure it is well-formed. If `/schema` is specified, `/validate` is implicitly specified. If the XML document does not conform to the DTD or schema, validation fails, error `-1504` is returned, and the `struct` output parameter is left unchanged..
`/schema`	Include schema information, if present, during conversion and validate the XML against it. By default this information is ignored. See Conversion with /schema and Data Types.

Parameters

Table: Parameters

Parameter	Data Type	Description
StructTarget	`struct` or `any`	Variable, parameter, or non-database field to hold the generated Struct .
XmlDocument	String	XML document to parse; value can be a string, file name, or a URL. The XML document can only contain one XML root element. The format of the parameter determines the type. A string must begin with the open tag angle bracket (<), and a URL must have the format *Protocol://Address*. Otherwise, the parameter is interpreted as a file name.
SchemaList	String	Namespace and location of one or more schemas. For a single schema, *SchemaList* can be a file name, URL, or a Uniface index list. For multiple schemas, it must be a Uniface list of name-value pairs in the format: {`NAMESPACE=` *Namespace* `;` }`LOCATION=` *Location* `NAMESPACE` cannot exceed 4096 characters; `LOCATION=`cannot exceed 260 characters.

Return Values

`0` is returned in `$status` if the conversion was successful. However, non-fatal errors may have occurred during conversion, because everything that is not recognized or usable is ignored. Warnings about such conditions are made available in `DETAIL` sublist of `$procReturnContext`. See $procReturnContext for xmlToStruct.

Table: Common Errors Returned in `$procerror` after `xmlToStruct`

Error	Error Constant	Meaning
`-4`	`<IOSERR_OPEN_FAILURE>`	Specified source does not exist or cannot be opened. Or, a specified URL is invalid or cannot be read; Or, the `NAME` or `LOCATION` are too long.
`-13`	`UIOSERR_OS_COMMAND`	The specified file name specified by XmlDocument is too long.
`-140`	`<UPROCERR_OPERAND>`	The Struct Target is not a variable or parameter of type `struct` or `any`.
-1503	`<UXMLERR_PARSE>`	The XML source is not well-formed XML.
`-1504`	`< UXMLERR_VALIDATE>`	The XML source does not conform to the DTD.

Use

Allowed in all component types.

On iSeries, there is no schema support, so `/schema` and *SchemaList* cannot be used.

Description

Use the `xmlToStruct` statement to transform an XML document to a Struct. For example, this Struct:

- Always contains a nameless root Struct that represents the XML document. It can optionally hold XML declaration tags.
- The XML root element is always the first named child struct of this nameless struct.

Thus, `<movie>The Matrix</movie>` is converted to the following Struct:

```
[]
 [movie] = "The Matrix"
```

> **Note:** `xmlToStruct` cannot be used to convert XML snippets.

An XML document always has a root node and it may have constructs that appear at the same level as the root node, such as the XML declaration, the DOCTYPE declaration, and comments. Some of these constructs are handled as annotations but others may be treated as Structs if `/full` is used.

During conversion `xmlToStruct`, transforms the XML document to one Struct at the top level, which holds the root node and its sibling constructs. In addition to converting XML constructs to Struct nodes, `xmlToStruct` sets annotations in `$tags` Struct of each Struct node.

Example: Viewing the Struct

This can be clearly seen in the string representing the Struct that is returned by `$dbgString`, which can be used during development to view and analyze the Struct. For example:

```
; INPUT_FLD is xmlstream
; OUTPUT_FLD is string
; $InputStruct$ is struct component variable
xmlToStruct /full $InputStruct$, INPUT_FLD
OUTPUT_FLD = "%%($InputStruct$->$dbgString)%%%"
```

If INPUT_FLD contains:

```
<?xml version="1.0" encoding="UTF-8"?>
<div class="note">Text can be <b>bold</b> or <em>italic</em></div>
```

the following is put into OUTPUT_FLD:

```
[]
 [div]
 [$tags]
 [xmlClass] = element
 Text can be
 [b] = bold
 [$tags]
 [xmlClass] = element
 or
```

```
[em] = italic
[$tags]
[xmlClass] = element
```

For each XML construct, there is a set of annotations that may be set, depending on the XML contents.

Example: Getting Information About Structs

Other Struct functions enable you to access and manipulate the Struct members in ProcScript.

For example:

```
$1 = $InputStruct$->$tags->xmlEncoding
```

Result: `$1 = "UTF-8"`

```
$2 = $InputStruct$->$collSize
```

Result: `$2 = 1`, meaning this Struct contains only one member. The `$tags` Struct is not counted.

Conversion with `/full`

By default XML comments, DOCTYPE declaration, and namespace declarations are ignored when converting to Structs.

To include them, use the `/full` switch. This ensures that all XML constructs, including comments, doctype declaration, and namespace declarations (if present), are also converted to Structs. This is critical if you need to reconstruct an equivalent XML document from the Struct.

> **Note:** The use of `/full` does not imply `/schema` or `/whitespace`.

If `/full` is specified, and `<![CDATA[...]]>` occurs within the character data, it is converted to a separate nameless scalar Struct with the `xmlClass` tags set to `CDATA`. Otherwise, the `<![CDATA[...]]>` piece is incorporated into one scalar Struct for the whole character value; it becomes indistinguishable from the rest of the data.

Namespace Handling

Qualified names are those consisting of namespace URI that qualify a local name. For example, the local name `schema` can be qualified by the namespace `"http://www.w3.org/2001/XMLSchema"` which together provide a unique identifier.

Namespaces can be defined with an alias. For example, `xmlns:s="http://www.w3.org/2001/XMLSchema"` means that within the scope of this definition `s` is the namespace `http://www.w3.org/2001/XMLSchema"`.

An alias can be used in a qualified name. For example, `s:schema` and its semantics are completely defined by the XML Schema Definition and should not be confused with any other meaning `schema`.

By default, `xmlToStruct` recognizes qualified names only in element and attribute names, and records the namespace declaration in the `$tags` members of the element or attribute Struct node.

When `xmlToStruct /full` is used, it also recognizes:

- Namespace declarations that are not actually used in a qualified name.
- Qualified names that are used as values of elements and attributes. This type of construction is used in specialised XML documents that describe or manipulate other XML documents.

For example, in the following XML namespace definition:

```
<ns1:element ❶ xmlns:ns1="www/namespace/example1" ❷
             xmlns:ns2="www/namespace/example2" ❸ ns2:attribute=a_value> ❹
```

1. `element` is a local name within the `ns1` namespace.
2. `ns1` is an alias or abbreviation of the namespace `www/namespace/example1`.
3. `ns2` is an alias or abbreviation of the namespace `www/namespace/example2`.
4. `attribute` is a local name within the `ns2` namespace.

By default, `xmlToStruct` registers this in the Struct node as:

```
[element] ❶
    [$tags]
        [xmlClass]=element
        [xmlNamespaceURI]=www/namespace/example1 ❷
        [xmlNamespaceAlias]=ns1
    [attribute]="a_value"
        [$tags]
            [xmlClass]=attribute ❹
            [xmlNamespaceURI]=www/namespace/example2 ❸
            [xmlNamespaceAlias]=ns2
```

The default behaviour records only the namespace declarations that are actually used, so if there were a third namespace declaration such as `xmlns:ns3="www/namespace/example3"`, the Struct nodes would not contain it because it is not used in those nodes.

In the following XML example, the attribute contains a namespace definition as a value:

```
<ns1:element

 ns2:attribute=”ns3:a_value">
```

By default, `xmlToStruct` produces:

```
[element]
 [$tags]
 [xmlClass]=element
 [xmlNamespaceURI]=www/namespace/example1
 [xmlNamespaceAlias]=ns1
 [attribute]="ns3:a_value"
 [$tags]
 [xmlClass]=attribute
 [xmlNamespaceURI]=www/namespace/example2
```



```
 [xmlNamespaceAlias]=ns2
```

Notice that the value of the attribute is `ns3:a_value`, but there is no definition of `ns3`.

In these cases, you must use the `/full` qualifier to ensure that all of the original namespace declarations are recorded in the Struct nodes. Thus `xmlToStruct/full` produces:

```
[element]
 [$tags]
 [xmlClass]=element
 [xmlNamespaceURI]=www/namespace/example1
 [xmlNamespaceAlias]=ns1
 [ns1]="www/namespace/example1"
 [$tags]
 [xmlClass]=namespace-declaration
 [ns2]="www/namespace/example2"
 [$tags]
 [xmlClass]=namespace-declaration
 [ns3]="www/namespace/example3"
 [$tags]
 [xmlClass] = namespace-declaration
 [attribute]="ns3:a_value"
 [$tags]
 [xmlClass]=attribute
 [xmlNamespaceURI]=www/namespace/example2
 [xmlNamespaceAlias]=ns2
```

Conversion with `/schema`

If `/schema` is used, the specified schema (or schemas) determine which whitespace is significant and which is not.

If schemas are specified, each Namespace+Location pair in the *SchemaList* is added to the list of schema locations as if they were specified in the XML source itself using the `schemaLocation=` keyword.

There can be only one location without accompanying namespace in the list, because the XML parser supports only one location specified by the `noNamespaceSchemaLocation` keyword.

For example:

- Index list has one item, a schema location:

  ```
  xmlToStruct vStruct, "file.xml", "schema.xsd"
  ```

- Index list has one item, an associated list with only a schema location; this is equivalent to the previous command:

  ```
  xmlToStruct vStruct, "file.xml", "LOCATION=schema.xsd"
  ```

When multiple schemas are specified, Namespace + Location pairs must be passed as an associated list. For example:


```
vSchemaList = "" ; initialize outer list
vItems = "" ; initialize sublist
; create Namespace+Location sublist
putitem/id vItems, "NAMESPACE", "http://www.shiporder.com"
putitem/id vItems, "LOCATION", "shiporder.xsd"
putitem vSchemaList, -1, vItems ; add items list to outer list
; add a second pair:
vItems = ""
putitem/id vItems, "NAMESPACE", "http://www.customs.com"
putitem/id vItems, "LOCATION", "custom.xsd"
putitem vSchemaList, -1, vItems
xmlToStruct vStruct, "file.xml", vSchemaList
```

Data Types

If `/schema` is not specified, the Uniface data type of every element and attribute is `string`.

If the `/schema` switch is specified, elements and attributes with predefined data types are included in the Struct as scalar Structs of matching Uniface data types. The values themselves are converted to Uniface format.

> **Note:** Supplying a schema list in the third parameter does not imply that schema information is to be included; the parameter may be supplied just to satisfy `/validate`, not because you want the schema information.

Table: XML - Uniface Data Type Mapping

Uniface	XML Data Types	Remarks
`string`	string, normalized string, token, language, name, NCName, token, NMTOKENS, ID, IDREF, IDREFS, ENTITY, ENTITIES	
	duration, gYearMonth, gYear, gMonthDay, gDay, gMonth	There is no equivalent in Uniface for these XML data types.
`date`, `time` , `datetime`	date, time, dateTime	If the value has the Z or [+|-]hh:mm timezone suffix, that timezone and the local machine timezone are used to calculate the corresponding time in the local timezone.
`numeric`	decimal, integer, long, int, short, byte nonPositiveInteger, negativeInteger nonNegativeInteger, positiveInteger, unsignedLong, unsignedInt, unsignedShort, unsignedByte	
`boolean`	boolean	The value is `T` or `F`.

Uniface	XML Data Types	Remarks
`float`	float, double	If the special values:`INF`, `-INF` or `NaN` are encountered, it is a string with that value .
`raw`	base64Binary, hexBinary	Value is the result of the base64-to-raw and hex-to-raw conversion performed by `xmlToStruct`.
	NULL	An element can be declared with the attribute `nillable="true"` in a schema. In an XML document that has the value of an element set to NULL, this is done by adding the attribute `xs:nil="true"` to the element. The resulting Struct contains a member named `nil` with `xmlClass="attribute"` for each element whose value is NULL. The value of the struct member itself will be an empty string.

$procReturnContext for xmlToStruct

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

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

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

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

`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:

Item	Description
	USTRUCTERR_TAGVALUE_NOT_APPLICABLE—Annotation xmlClass has an unknown or illegal value (based on the current context).
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.
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.
TAGVALUE	Value of the tag specified by TAGNAME.
EXPECTED	Expected object for the context: Struct valid on XML document level Struct valid on XML element level DTD declaration DTD attribute declaration

Related concepts

- Transforming Complex Data Using Structs
- Structs for XML Data
- Struct Annotations for XML
- $tags
- structToXml
- Uniface XML Constructs