PARSE XML

PARSE XML *operand1* [**INTO** [**PATH** *operand2*] [**NAME** *operand3*] [**VALUE** *operand4*]]
[[**NORMALIZE**] **NAMESPACE** *operand5* **PREFIX** *operand6*]
statement...
END-PARSE *(structured mode only)*
[**LOOP**] *(reporting mode only)*

This chapter covers the following topics:

- Function
- Syntax Description
- Examples

For an explanation of the symbols used in the syntax diagram, see *Syntax Symbols*.

Belongs to Function Group: *Internet and XML*

Function

The `PARSE XML` statement allows you to parse XML documents from a Natural program. See also *Statements for Internet and XML Access* in the *Programming Guide*.

It is recommended that you use dynamic variables when using the `PARSE` statement, because it is impossible to determine the length of a static variable. Using static variables could in turn lead to the truncation of the value that is to be written into the variable.

For information on Unicode support, see `PARSE XML` in the *Unicode and Code Page Support* documentation.

Mark-Up

The following are markings used in path strings to represent the different data types in an XML document (on ASCII-based systems):

Marking	XML Data	Location in Path String
?	Processing instruction (except for `<?XML...?>`)	end
!	Comment	end
C	CDATA section	end
@	Attribute (on mainframes: § or @, depending on session code page and terminal emulation)	before the attribute name
/	Closing tag and/or parent name separator in a path	end or between parent names
$	Parsed data - character data string	end

By using this additional markup in the path string, one can more easily identify the different elements of the XML document in the output document.

Global Namespace

To specify the global namespace, use a colon (:) as prefix and an empty URI.

Related System Variables

The following Natural system variables are automatically created for each `PARSE XML` statement issued:

- `*PARSE-TYPE`
- `*PARSE-LEVEL`
- `*PARSE-ROW`
- `*PARSE-COL`
- `*PARSE-NAMESPACE-URI`

The notation `(r)` after `*PARSE-TYPE`, `*PARSE-LEVEL`, `*PARSE-ROW`, `*PARSE-COL` and `*PARSE-NAMESPACE-URI` is used to indicate the label or statement number of the statement in which the `PARSE` was issued. If `(r)` is not specified, the corresponding system variable represents the system variable of the XML data currently being processed in the active `PARSE` processing loop.

For more information on these system variables, see the *System Variables* documentation.

Syntax Description

Operand Definition Table:

Operand	Possible Structure					Possible Formats													Referencing Permitted	Dynamic Definition
operand1	C	S				A	U	B											yes	no
operand2		S				A	U	B											yes	yes
operand3		S				A	U	B											yes	yes
operand4		S				A	U	B											yes	yes
operand5		S	A			A	U	B											yes	yes
operand6		S	A			A	U	B											yes	yes

Syntax Element Description:

operand1	*operand1* represents the XML document in question. The XML document may not be changed while it is being parsed. If you try to change the XML document during parsing (by writing into it, for example), an error message will be displayed.
operand2	*operand2* represents the `PATH` of the data in the XML document. The `PATH` contains the name of the identified XML part, the names of all parents, as well as the type of the XML part. **Note:** The information given with `PATH` can be used to easily fill a tree view. See also *Example 1 - Using operand2*.
operand3	*operand3* represents the `NAME` of a data element in the XML document. If `NAME` has no value, then the dynamic variable associated with it will be set to `*length()=0`, which is a static variable filled with a blank. See also *Example 2 - Using operand3*.
operand4	*operand4* represents the content (`VALUE`) of a data element in the XML document. If there is no value, a given dynamic variable will be set to `*length()=0,` which is a static variable filled with a blank. See also *Example 3 - Using operand4*.

operand5* and *operand6 **NORMALIZE NAMESPACE** **PREFIX**	The NAMESPACE URI or Uniform Resource Identifier (*operand5*) and the namespace PREFIX (*operand6*) are copied during runtime. Therefore, modifying the namespace mapping arrays inside the PARSE XML loop will not affect the parser.

	operand5 and *operand6* are one-dimensional arrays with an equal number of occurrences.
	Namespace normalization is a feature of the PARSE statement. XML is capable of defining namespaces for the element names:
	`<myns:myentity xmlns:myns="http://myuri" />`
	The NAMESPACE definition consists of two parts:
	● a namespace PREFIX (which is, in this case, myns) and
	● a URI (myuri) to define the namespace.
	The namespace PREFIX is part of the element name. This means, that for the PARSE statement, and especially for *operand2*, the generated PATH strings depend on the namespace PREFIX. If the path inside a Natural program is used to indicate specific tags, then this will fail if an XML document uses the correct NAMESPACE (URI), but with a different PREFIX.
	With namespace normalization, all namespace PREFIXes can be set to defaults which have been defined in the NAMESPACE clause. The first entry will be the one used if a URI is specified more than once. If more than one PREFIX is used in the XML document, then only the first one will be taken into account for the output. The rest will be ignored.
	The NAMESPACE clause contains pairs of namespace URIs and prefixes. For example:
	`uri(1) := ’http://namespaces.softwareag.com/natural/demo’`
	`pre(1) := ’nat:’`
	If NAMESPACE is defined inside an XML document, the parser checks to see if that namespace (URI) exists in the normalization table. The prefix of the normalization table is used for all output data from the PARSE statement, instead of the namespace defined in the XML document.
	See also:
	● *Example 4 - Using operand5 and operand6*
	● *Example 5 - Using operand5 and operand6 with Namespace Normalization*

	Additional Information Concerning PREFIX: In addition, the following applies to the prefix definition: ● The prefix definition in the namespace normalization array always has to end in a colon (:), since this is the string that will be replaced. ● A PREFIX or a URI may only occur once in a namespace normalization array. ● If a PREFIX or the NAMESPACE URI contains trailing blanks (e.g. when using a static variable), the trailing blanks will be removed before the external parser is called. ● If the PREFIX definition at the namespace normalization only contains a colon (:), then the NAMESPACE PREFIX will be deleted.

Examples

- Example 1 - Using operand2
- Example 2 - Using operand3
- Example 3 - Using operand4
- Example 4 - Using operand5 and operand6
- Example 5 - Using operand5 and operand6 with Namespace Normalization

Example 1 - Using *operand2*

The following XML code

```
myxml := '<?xml version="1.0"  ?>'-
         '<employee personnel-id="30016315" >'-
         '<full-name>'-
         '<!--this is just a comment-->'-
         '<first-name>RICHARD</first-name>'-
         '<name>FORDHAM</name>'-
         '</full-name>'-
         '</employee>'
```

processed by the following Natural code:

```
PARSE XML myxml INTO PATH mypath
  PRINT mypath
END-PARSE
```

produces the following output:

```
employee
employee/@personnel-id
employee/full-name
employee/full-name/!
employee/full-name/first-name
employee/full-name/first-name/$
employee/full-name/first-name//
employee/full-name/name
employee/full-name/name/$
employee/full-name/name//
employee/full-name//
employee//
```

Example 2 - Using *operand3*

The following XML code

```
myxml := '<?xml version="1.0"  ?>'-
         '<employee personnel-id="30016315" >'-
         '<full-name>'-
         '<!--this is just a comment-->'-
         '<first-name>RICHARD</first-name>'-
         '<name>FORDHAM</name>'-
         '</full-name>'-
         '</employee>'
```

processed by the following Natural code:

```
PARSE XML myxml INTO PATH mypath NAME myname
  DISPLAY (AL=39) mypath myname
END-PARSE
```

Note:
produces the following output:

```
               MYPATH                                MYNAME
----------------------------------    -----------------------------------


employee                              employee
employee/@personnel-id                personnel-id
employee/full-name                    full-name
employee/full-name/!
employee/full-name/first-name         first-name
employee/full-name/first-name/$
employee/full-name/first-name//       first-name
employee/full-name/name               name
employee/full-name/name/$
employee/full-name/name//             name
employee/full-name//                  full-name
employee//                            employee
```

Example 3 - Using *operand4*

The following XML code

```
myxml := ’<?xml version="1.0"  ?>’-
         ’<employee personnel-id="30016315" >’-
         ’<full-name>’-
         ’<!--this is just a comment-->’-
         ’<first-name>RICHARD</first-name>’-
         ’<name>FORDHAM</name>’-
         ’</full-name>’-
         ’</employee>’
```

processed by the following Natural code:

```
PARSE XML myxml INTO PATH mypath VALUE myvalue
  DISPLAY (AL=39) mypath myvalue
END-PARSE
```

produces the following output:

```
               MYPATH                                MYVALUE
----------------------------------    -----------------------------------


employee
employee/@personnel-id                30016315
employee/full-name
employee/full-name/!                  this is just a comment
employee/full-name/first-name
employee/full-name/first-name/$       RICHARD
employee/full-name/first-name//
employee/full-name/name
```

```
employee/full-name/name/$             FORDHAM
employee/full-name/name//
employee/full-name//
employee//
```

Example 4 - Using *operand5* and *operand6*

The following XML code

```
myxml := ’<?xml version="1.0"  ?>’-
         ’<nat:employee nat:personnel-id="30016315"’-
         ’ xmlns:nat="http://namespaces.softwareag.com/natural/demo">’-
         ’<nat:full-Name>’-
         ’<nat:first-name>RICHARD</nat:first-name>’-
         ’<nat:name>FORDHAM</nat:name>’-
         ’</nat:full-Name>’-
         ’</nat:employee>’
```

processed by the following Natural code:

```
PARSE XML myxml INTO PATH mypath
  PRINT mypath
END-PARSE
```

produces the following output:

```
nat:employee
nat:employee/@nat:personnel-id
nat:employee/@xmlns:nat
nat:employee/nat:full-Name
nat:employee/nat:full-Name/nat:first-name
nat:employee/nat:full-Name/nat:first-name/$
nat:employee/nat:full-Name/nat:first-name//
nat:employee/nat:full-Name/nat:name
nat:employee/nat:full-Name/nat:name/$
nat:employee/nat:full-Name/nat:name//
nat:employee/nat:full-Name//
nat:employee//
```

Example 5 - Using *operand5* and *operand6* with Namespace Normalization

Using `NORMALIZE NAMESPACE`, the same XML document as in Example 4 with a different `NAMESPACE PREFIX` would produce exactly the same output.

XML code:

```
myxml := ’<?xml version="1.0"  ?>’-
         ’<natural:employee natural:personnel-id="30016315"’-
         ’ xmlns:natural="http://namespaces.softwareag.com/natural/demo">’-
         ’<natural:full-Name>’-
         ’<natural:first-name>RICHARD</natural:first-name>’-
         ’<natural:name>FORDHAM</natural:name>’-
         ’</natural:full-Name>’-
         ’</natural:employee>’
```

Natural code:

```
uri(1) := ’http://namespaces.softwareag.com/natural/demo’
pre(1) := ’nat:’
*
PARSE XML myxml INTO PATH mypath NORMALIZE NAMESPACE uri(*) PREFIX pre(*)
  PRINT mypath
END-PARSE
```

Output of above program:

```
nat:employee
nat:employee/@nat:personnel-id
nat:employee/@xmlns:nat
nat:employee/nat:full-Name
nat:employee/nat:full-Name/nat:first-name
nat:employee/nat:full-Name/nat:first-name/$
nat:employee/nat:full-Name/nat:first-name//
nat:employee/nat:full-Name/nat:name
nat:employee/nat:full-Name/nat:name/$
nat:employee/nat:full-Name/nat:name//
nat:employee/nat:full-Name//
nat:employee//
```