
CHAPTER 1 *XPath*

This chapter has information you may need to use XPath arguments to various *Janus SOAP* \$functions. Most of the information is taken from the XPath standard, which is the authoritative reference:

<http://www.w3.org/TR/xpath>

1.1 XPath operation

The purpose of XPath is to select a subset of nodes from a document; this is done in version 6.4 of *Janus SOAP* using a Location Path, described by the `LocPath` production. A Location Path consists of a series of Steps (`Step` ([4]) production). Each Step operates by taking an input set of nodes from the preceding step, and creating an output set of nodes. The output of the last step is the set of nodes selected by the XPath expression.

An example XPath expression is:

`pitm[2]/partnum`

This expression contains two Steps (the slash symbol (/) is used to separate the Steps in a Location Path.).

Often a Step will start with an element name, which selects all the child elements with that name. In the above example, `partnum` children of `pitm` elements are selected. These *child* relationships are one kind of relationship between the input to a Step and the first part of the algorithm; the kinds of relationships are shown in the `AxisName` ([6]) production.

The element names in the above example are a form of Node Test, described by the `NodeTest` ([7]) production; the Node Test is used to restrict the set of nodes.

Square brackets (“[...]”) in a Step surround another form of restriction, which is called a Predicate, given by the production ([8]) with the same name. A Predicate is a much more open-ended type of restriction, allowing various functions and operations, including Booleans.

The operation of a Step is as follows:

1. A Step consists of an Axis, Node Test, and zero or more Predicates.
2. The input to a Step is a set of context nodes.
3. The Axis produces sets of nodes, one set for each context node.
4. Each of these sets is filtered by the Node Test.
5. Each of the resulting sets is filtered by a Predicate.
6. The final filtered sets are combined (using set union), and the result is the set of nodes which becomes input to the following Step.
7. The result of the final Step is the result of the Location Path.

1.1.1 Axes

The various forms of the `AxisName` ([6]) production generate nodes based on a context node using the simple tree relationships described by the name. For example, `attribute::` (abbreviated `@`) generates the set of all attributes of a context node.

1.1.2 Node tests

The various forms of the `NodeTest` ([7]) production filter nodes as follows:

NodeType '(' ')'

This selects any node that has the respective node type, for example `comment()` selects all `Comment` nodes in a node set.

'processing-instruction' '(' `Lit` ')'

This selects any Processing Instruction node if the target name is equal to the value of `Lit`.

The other forms test the **name** of a node, after restricting the type of node depending on the Axis:

- Name tests in the `attribute::` Axis restrict to Attribute nodes.
- Name tests in any other Axis restrict to Element nodes.

The name tests then filter the resulting nodes as follows:

'*' This selects a node of the selected type with any name.

QName This selects a node of the selected type which has the same name as `QName`.

1.1.3 Predicates

Each node set which is the result of Node Test filtering is input to the series of Predicates in the Step; each Predicate's result sets are passed to the following one, with union of the results of the last Predicate (or the Node Test, if there are no Predicates) forming the result of the Step.

There is a variety of Predicates, and except for a numeric Predicate, a Predicate selects a node if the value of the Predicate, converted to a Boolean, is `true`.

Three common forms of predicates are:

1. A predicate which is a Location Path expression, for example, an element name such as `expired-date`, selects a node if the result of that Location Path, using the node as its context node, is non-empty.
2. A predicate which is a Location Path expression and comparison operator and literal, such as `Price > "200"`, selects a node if any node in the result of that Location Path holds the specified relationship to the literal.
3. If a Predicate p is numeric, it is equivalent to the expression `position()=p`.

The `position()` function returns the position in the set of the node which the predicate is filtering.

1.2 XPath syntax used in version 6.4 *Janus SOAP*

This section contains a condensed excerpt of the XPath syntax, showing only those parts of XPath used by *Janus SOAP* in version 6.4.

```
[1] LocPath ::= RelativeLocPath
           | '/' RelativeLocPath?

[3] RelativeLocPath ::= Step ('/' Step)*

[4] Step ::= '.' /* self::node() */
         | '..' /* parent::node() */
         | (AxisName '::' | '@')? NodeTest Predicate*

[6] AxisName ::= 'attribute' | 'child'
            | 'parent' | 'self'

[7] NodeTest ::= '*' | QName | 'node()'
            | 'comment()' | 'text()'
            | 'processing-instruction(' Lit? ')'
```

[8] Predicate ::= '[' PredExpr '']

PredExpr ::= PositiveInteger
| PathExpr /* "Existence test" */
| Comparison

Comparison ::= PathExpr
('=' | '!=' | '<' | '>' | '<=' | '>=') Lit

[29] Lit ::= ''' [¬"]* '''
| """ [¬']* """

[30] PositiveInteger ::= [1-9] [0-9]*

Notes:

- When @ is used in a Step ([4]), it is an abbreviation for

attribute::

- The syntax for Step ([4]) notes that it may begin directly with a NodeTest; in that case, `child::` is implied before the NodeTest. From the XML Namespaces Recommendation we get this syntax for QName:

[NA4] NCName ::= (Letter | '_') (NCNameChar)*
[NB5] NCNameChar ::= Letter | Digit | '.' | '-' | '_'
[NC6] QName ::= (NCName ':')? NCName

- A node is selected by a PositiveInteger predicate if the position of the node, in the set which the predicate is filtering, is equal to that PositiveInteger.
- A node is selected by an Existence test if the result of the PathExpr, using that node as the context node, is non-empty.
- A node is selected by a Comparison if any node in the result of the PathExpr, using that node as the context node, holds the specified relationship to the Lit.

Example Document - Serialized View

```

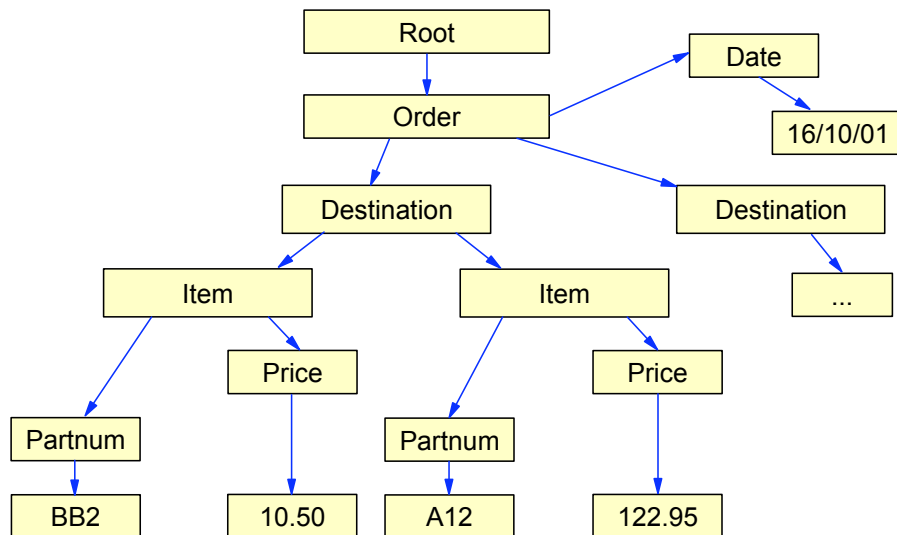
<Order>
  <Date>16/10/01</Date>
  <Destination>
    <Item>
      <Partnum>BB2</Partnum>
      <Price>10.50</Price>
    </Item>
    <Item>
      <Partnum>A12</Partnum>
      <Price>122.95</Price>
    </Item>
  <Destination>...</Destination>
</Order>

```



Sirius Software, Inc.

Example Document - Tree View



Sirius Software, Inc.